TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

KHOA CÔNG NGHỆ THÔNG TIN

PHẠM DUY KHÁNH - 522H0064

PHẠM VĂN PHÚC - 522H0068

NGUYỄN HUỲNH ANH KHOA - 522H0046

TỐNG NGUYÊN GIA HUY -522H0077

ĐỒ ÁN CUỐI KÌ

WEBSITE THƯƠNG MẠI ĐIỆN TỬ

BÁO CÁO CUỐI KỲ

MẪU THIẾT KẾ

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2025

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

KHOA CÔNG NGHỆ THÔNG TIN

PHẠM DUY KHÁNH - 522H0064

PHẠM VĂN PHÚC - 522H0068

NGUYỄN HUỲNH ANH KHOA - 522H0046

TỐNG NGUYÊN GIA HUY -522H0077

ĐỒ ÁN CUỐI KÌ

WEBSITE THƯƠNG MẠI ĐIỆN TỬ

BÁO CÁO CUỐI KỲ

MẪU THIẾT KẾ

Người hướng dẫn

Thầy Hà Lê Hoài Trung

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2025

LỜI CẢM ƠN

Chúng em xin chân thành cảm ơn ……
……
……
……
……
……
……
……

TP. Hồ Chí Minh, ngày ... tháng ... năm 20..

Tác giả

(Ký tên và ghi rõ họ tên)

CÔNG TRÌNH ĐƯỢC HOÀN THÀNH
TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi và được sự hướng dẫn khoa học của Thầy Hà Lê Hoài Trung. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong Dự án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung Dự án của mình. Trường Đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày ... tháng ... năm 20..

Tác giả

(Ký tên và ghi rõ họ tên)

Phạm Duy Khánh

Phạm Văn Phúc

Nguyễn Huỳnh Anh Khoa

Tống Nguyễn Gia Huy

ĐỒ ÁN CUỐI KÌ

TÓM TẮT

Dự án sẽ tập trung triển khai một hệ thống thương mại điện tử với các chức năng cốt lõi như quản lý người dùng, sản phẩm, giỏ hàng và thanh toán. Mỗi chức năng sẽ được phân tích và áp dụng mẫu thiết kế phù hợp như Singleton, Factory, Strategy, Observer,… nhằm đảm bảo tính mở rộng, tái sử dụng và dễ bảo trì. Việc sử dụng class diagram giúp minh họa rõ ràng cấu trúc và mối quan hệ giữa các lớp trong từng mẫu thiết kế. Qua đó, dự án không chỉ củng cố kiến thức lý thuyết mà còn rèn luyện kỹ năng áp dụng vào thực tiễn phát triển phần mềm.

MỤC LỤC

DANH MỤC HÌNH VẼ

CHƯƠNG 1. GIỚI THIỆU BÀI TOÁN

1.1 Mô tả chung

Trang web này là một nền tảng thương mại điện tử được thiết kế để cung cấp một trải nghiệm mua sắm trực tuyến đơn giản, tiện lợi và an toàn. Người dùng có thể duyệt qua các sản phẩm, thêm chúng vào giỏ hàng, thực hiện thanh toán và theo dõi trạng thái đơn hàng của mình chỉ với vài thao tác. Trang web hỗ trợ nhiều loại sản phẩm từ các ngành hàng khác nhau, giúp người dùng dễ dàng tìm kiếm và mua sắm mọi lúc, mọi nơi.

Để đảm bảo hiệu suất, tính mở rộng và sự linh hoạt trong việc quản lý các chức năng quan trọng như giỏ hàng, thanh toán và thông báo đơn hàng, trang web áp dụng các mẫu thiết kế phần mềm (Design Patterns). Các mẫu thiết kế này giúp tối ưu hóa quá trình phát triển, dễ dàng bảo trì và mở rộng hệ thống khi cần thiết. Các mẫu thiết kế được áp dụng bao gồm Singleton, Strategy, Factory Method, Observer và Template Method, mang lại sự ổn định và hiệu quả cao cho hệ thống.

1.2 Danh sách chức năng

1.2.1 Người dùng

- Đăng ký
- Đăng nhập
- Tìm kiếm sản phẩm
- Xem chi tiết sản phẩm
- Thêm sản phẩm vào giỏ hàng
- Quản lý giỏ hàng
- Thanh toán
- Quản lý đơn hàng
- Xuất hoá đơn
- Đánh giá sản phẩm

1.2.2 Quản lý (Admin)

- Quản lý người dùng
- Quản lý sản phẩm
- Quản lý loại sản phẩm
- Quản lý nhà cung cấp
- Quản lý đơn hàng
- Quản lý hoá đơn
- Xuất hoá đơn

1.3 Các chức năng chính

1.3.1 Tạo sản phẩm mới

Chức năng này cho phép người dùng có vai trò quản trị (admin) thêm sản phẩm vào hệ thống để hiển thị trong danh sách sản phẩm cho khách hàng. Các bước thực hiện bao gồm:

- **Truy cập giao diện quản trị**: Admin đăng nhập vào hệ thống và vào mục quản lý sản phẩm.
- **Nhập thông tin sản phẩm**: Admin cung cấp các thông tin chi tiết của sản phẩm, bao gồm:
    - **Tên sản phẩm**: Tên gọi ngắn gọn, dễ nhận diện.
    - **Mô tả sản phẩm**: Thông tin chi tiết về sản phẩm (chất liệu, kích thước, công dụng, v.v.).
    - **Giá bán**: Giá niêm yết của sản phẩm (hỗ trợ định dạng tiền tệ).
    - **Hình ảnh minh họa**: Tải lên hình ảnh sản phẩm (hỗ trợ các định dạng như JPG, PNG).
    - **Loại sản phẩm**: Phân loại sản phẩm theo danh mục (ví dụ: quần áo, điện tử, thực phẩm, v.v.).
    - **Số lượng tồn kho**: Số lượng sản phẩm có sẵn trong kho.
    - **Thông số chi tiết**: Thể hiện những thông số cho các thuộc tính riêng biệt cho từng loại sản phẩm.

- **Xác nhận và lưu**: Sau khi nhập đầy đủ thông tin, admin xác nhận để thêm sản phẩm vào cơ sở dữ liệu. Hệ thống kiểm tra tính hợp lệ của dữ liệu (ví dụ: giá không âm, hình ảnh đúng định dạng) trước khi lưu.
- **Thông báo kết quả**: Hệ thống hiển thị thông báo thành công hoặc lỗi nếu có vấn đề trong quá trình tạo sản phẩm.
- **Cập nhật danh sách sản phẩm**: Sản phẩm mới sẽ được hiển thị ngay lập tức trong danh sách sản phẩm trên giao diện người dùng sau khi tạo thành công.

Chức năng này đảm bảo admin có thể dễ dàng bổ sung sản phẩm mới với thông tin đầy đủ, hỗ trợ việc quản lý danh mục sản phẩm hiệu quả.

1.3.2 Quản lý giỏ hàng

Chức năng này giúp người dùng dễ dàng quản lý các sản phẩm đã chọn để mua. Người dùng có thể:

- **Thêm sản phẩm vào giỏ hàng:** Khi duyệt qua danh sách sản phẩm, người dùng có thể chọn các sản phẩm muốn mua và thêm chúng vào giỏ hàng. Mỗi sản phẩm sẽ có thông tin chi tiết như tên, giá và hình ảnh minh họa để giúp người dùng đưa ra quyết định.
- **Chỉnh sửa số lượng sản phẩm:** Người dùng có thể cập nhật số lượng của mỗi sản phẩm trong giỏ hàng. Nếu muốn mua nhiều sản phẩm hơn, chỉ cần điều chỉnh số lượng và hệ thống sẽ tự động tính lại tổng giá trị đơn hàng.
- **Xóa sản phẩm khỏi giỏ hàng:** Nếu người dùng thay đổi ý định và muốn loại bỏ một sản phẩm khỏi giỏ hàng, họ có thể dễ dàng xóa sản phẩm đó. Hệ thống sẽ cập nhật lại giỏ hàng và tổng giá trị đơn hàng sau khi xóa sản phẩm.
- **Tiến hành thanh toán:** Sau khi kiểm tra lại giỏ hàng, người dùng có thể tiến hành thanh toán đơn hàng. Hệ thống sẽ yêu cầu người dùng chọn phương thức thanh toán, bao gồm chuyển khoản, thẻ tín dụng, hoặc ví điện tử. Sau khi thanh

toán thành công, đơn hàng sẽ được tạo ra và lưu trữ trong hệ thống với các thông tin liên quan như ngày đặt hàng, danh sách sản phẩm, và tổng giá trị.

Tất cả các thao tác với giỏ hàng của khách hàng đều được thực hiện trên **một giỏ hàng duy nhất**, đảm bảo tính đồng bộ và nhất quán trong suốt quá trình mua sắm. Mọi thay đổi như chỉnh sửa số lượng, xóa sản phẩm hay thanh toán đều được cập nhật và phản ánh ngay lập tức trong giỏ hàng, giúp khách hàng dễ dàng theo dõi và kiểm soát mọi thông tin liên quan đến đơn hàng của mình.

1.3.3 Thêm sản phẩm vào giỏ hàng

Chức năng này cho phép người dùng đã đăng nhập lựa chọn và thêm sản phẩm vào giỏ hàng để tiến hành đặt hàng

- Cho phép thêm vào giỏ hàng, đặt hàng.
- Sau khi đăng nhập, người dùng có thể truy cập vào danh sách sản phẩm được hiển thị rõ ràng theo từng loại.
- Hệ thống cung cấp tính năng tìm kiếm và lọc giúp người dùng dễ dàng chọn lựa sản phẩm mong muốn.
- Mỗi sản phẩm đều có thông tin chi tiết như tên, mô tả, giá và hình ảnh minh họa. Người dùng có thể thêm sản phẩm vào giỏ hàng, cập nhật số lượng hoặc xóa sản phẩm khỏi giỏ. Khi đã hoàn tất lựa chọn, người dùng có thể tiến hành đặt hàng.
- Sau khi người dùng thêm sản phẩm vào giỏ hàng thì số lượng sản phẩm trong giỏ hàng sẽ được cập nhật tức thời.

1.3.4 Thanh toán đa phương thức

Chức năng Thanh toán đa phương thức cho phép người dùng hoàn tất đơn hàng bằng cách chọn và xử lý thanh toán qua các phương thức khác nhau. Chức năng này được thiết kế dưới dạng mô phỏng để kiểm thử quy trình mà không cần tích hợp hệ thống thanh toán thực tế

- **Xử lý thanh toán:**
    - Hệ thống sử dụng các lớp PaymentRequest và PaymentResponse để mô phỏng quy trình thanh toán:
        - PaymentRequest: Tạo yêu cầu thanh toán với thông tin đơn hàng (danh sách sản phẩm, tổng giá trị, phương thức thanh toán).
        - Xác nhận đơn hàng: Hiển thị thông tin đơn hàng để người dùng kiểm tra lần cuối (sản phẩm, số lượng, tổng giá, phương thức thanh toán).
        - Xử lý giao dịch: Mô phỏng việc gửi yêu cầu đến cổng thanh toán và nhận phản hồi (thành công hoặc thất bại).
        - PaymentResponse: Trả về kết quả giao dịch (trạng thái, mã giao dịch nếu thành công).
- **Kết quả thanh toán:**
    - Sau khi xử lý, hệ thống hiển thị thông báo:
        - Nếu thành công: Cung cấp mã đơn hàng, trạng thái "Đã thanh toán", và thông tin đơn hàng.
        - Nếu thất bại: Hiển thị lý do lỗi và cho phép thử lại với phương thức khác.
    - Đơn hàng được cập nhật trạng thái trong cơ sở dữ liệu (ví dụ: từ "Chờ xử lý" sang "Đã thanh toán").
    - Giỏ hàng được làm trống sau khi thanh toán thành công.

1.4 Phân tích mẫu thiết kế cho bài toán

1.4.1 Singleton – Quản lý giỏ hàng

Mẫu thiết kế Singleton là một mẫu thiết kế thuộc nhóm Creational Patterns (mẫu tạo lập), được sử dụng để đảm bảo rằng một lớp chỉ có một thể hiện duy nhất (instance) trong toàn bộ ứng dụng và cung cấp một điểm truy cập toàn cục đến thể hiện đó.

Lý do sử dụng pattern này vì bản chất của Singleton là chỉ có một thể hiện duy nhất tồn tại xuyên suốt toàn bộ quá trình sử dụng và có khả năng truy cập toàn cục vậy. Vì vậy mẫu Singleton đảm bảo chỉ tồn tại một thể hiện duy nhất của giỏ hàng cho mỗi người dùng trong suốt phiên làm việc. Điều này giúp tránh tạo trùng giỏ hàng, giảm thiểu lỗi khi xử lý dữ liệu (ví dụ: thêm nhầm sản phẩm vào nhiều giỏ khác nhau).

Mô tả các thành phần:

- CartSingletonService (Subject - Singleton Class)
- CartController (Client)

Lợi ích:

- Truy cập toàn cục: Giỏ hàng có thể được lấy từ bất kỳ phần nào trong ứng dụng (trang sản phẩm, thanh toán, v.v).
- Đồng bộ trạng thái: Đảm bảo thông tin giỏ hàng (sản phẩm, số lượng...) luôn nhất quán, không bị lệch khi thao tác ở nhiều nơi.
- Giảm thiểu sử dụng tài nguyên: Vì chỉ có một thể hiện duy nhất, Singleton giúp tiết kiệm bộ nhớ và tài nguyên hệ thống so với việc tạo nhiều instance không cần thiết.

Hạn chế :

- **Khó kiểm tra**: Singleton có thể làm tăng độ phức tạp trong việc kiểm thử (unit testing) vì nó duy trì trạng thái toàn cục, gây khó khăn khi cố gắng mô phỏng các tình huống kiểm thử khác nhau.
- **Khó quản lý trong hệ thống đa người dùng**:
  Singleton chỉ phù hợp cho một người dùng trong một phiên làm việc. Trong ứng dụng đa người dùng, nếu không gắn Singleton với sessionId hoặc userId, tất cả người dùng có thể chia sẻ cùng một giỏ hàng, gây lỗi nghiêm trọng.
- **Phụ thuộc toàn cục (Global State)**:

Việc sử dụng Singleton tạo ra một trạng thái toàn cục, có thể dẫn đến sự phụ thuộc không mong muốn giữa các thành phần, làm mã nguồn khó bảo trì và mở rộng.

Tình huống ví dụ: Người dùng thêm sản phẩm từ nhiều trang khác nhau, sau đó tiến hành thanh toán – Singleton đảm bảo mọi thao tác đều áp dụng lên cùng một giỏ hàng duy nhất, tránh thất thoát hoặc trùng lặp.

1.4.2 Strategy – Thanh toán nhiều phương thức

Mẫu thiết kế Strategy là một mẫu thiết kế thuộc nhóm Behavioral Patterns (mẫu hành vi), được sử dụng để định nghĩa một tập hợp các thuật toán (chiến lược), đóng gói từng thuật toán trong một lớp riêng biệt và cho phép thay đổi thuật toán được sử dụng tại thời điểm chạy (runtime). Trong trường hợp này, Strategy được áp dụng để quản lý các phương thức thanh toán khác nhau (chuyển khoản, ví điện tử, thẻ tín dụng,…) trong một ứng dụng.

Lý do sử dụng:

- Linh hoạt tại runtime: Strategy cho phép ứng dụng chọn phương thức thanh toán phù hợp (ví dụ: chuyển khoản, ví điện tử, thẻ tín dụng) tại thời điểm người dùng thực hiện thanh toán, mà không cần sửa đổi logic cốt lõi của hệ thống.
- Dễ dàng mở rộng: Khi cần thêm phương thức thanh toán mới (ví dụ: thanh toán bằng tiền mã hóa), chỉ cần tạo một chiến lược mới mà không cần thay đổi mã nguồn hiện có, đảm bảo hệ thống dễ bảo trì và mở rộng.

Mô tả các thành phần:

- PaymentContext (Context)
- PaymentStrategy (Strategy Interface)
- BankTransfer, CreditCard, EWallet (Concrete Strategies)
- OrderService (Client - Controller)

Lợi ích:

- **Linh hoạt:** Hệ thống có thể dễ dàng thêm, xóa hoặc thay đổi các chiến lược thanh toán mà không làm ảnh hưởng đến các thành phần khác của ứng dụng.
- Tuân thủ nguyên lý OCP (Open/Closed Principle): Hệ thống mở để mở rộng (thêm chiến lược mới) nhưng đóng đối với sửa đổi (không cần thay đổi mã hiện có).
- **Tách biệt logic:** Mỗi chiến lược thanh toán được đóng gói trong một lớp riêng biệt, giúp mã nguồn rõ ràng, dễ hiểu, dễ kiểm thử (test) và bảo trì.
- Tái sử dụng: Các chiến lược có thể được tái sử dụng ở nhiều phần khác nhau của hệ thống hoặc trong các ứng dụng khác.

Hạn chế

- **Sự phức tạp trong quản lý**: Mặc dù dễ dàng mở rộng, nhưng nếu số lượng chiến lược thanh toán quá lớn, việc quản lý và duy trì chiến lược sẽ trở nên khó khăn, đặc biệt nếu chiến lược cần phải tương tác với nhiều phần khác nhau trong hệ thống
- **Có thể gây trùng lặp logic**: Các chiến lược thanh toán có thể có nhiều logic giống nhau, dẫn đến việc mã bị trùng lặp trong các lớp chiến lược.

Tình huống ví dụ: Người dùng chọn thanh toán bằng thẻ tín dụng, ví điện tử, hoặc chuyển khoản. Ứng dụng sẽ tự động chọn chiến lược phù hợp, và thực hiện xử lý tương ứng mà không cần thay đổi code gọi hàm.

1.4.3 Factory Method – Tạo sản phẩm theo loại

Mẫu thiết kế Factory Method là một mẫu thiết kế thuộc nhóm Creational Patterns (mẫu tạo lập), được sử dụng để định nghĩa một giao diện hoặc phương thức trừu tượng để tạo đối tượng, nhưng để các lớp con quyết định loại đối tượng cụ thể nào sẽ được tạo. Trong trường hợp này, Factory Method được áp dụng để tạo các sản phẩm khác nhau có các thuộc tính riêng biệt cho từng loại sản phẩm trong một ứng dụng thương mại điện tử.

Ngoài những thuộc tính chung cho tất cả các loại sản phẩm như tên sản phẩm, giá tiền, mô tả, nhà cung cấp, … Thì mỗi loại sản phẩm lại có những thuộc tính riêng biệt như:

- Laptop: CPU, RAM, bộ nhớ, …
- Tai nghe: Chống ồn, thời gian sử dụng, kết nối không dây, ..
- Điện thoại: Thời lượng pin, độ phân giải camera, kích thước màn hình, …
- Máy tính bảng: Kích thước màn hình, thời lượng pin, hỗ trợ bút thông minh, …
- Đồng hồ thông minh: Định vị GPS, thời gian sử dụng, khả năng chống nước …

Lý do sử dụng vì Factory Method giúp tạo ra các đối tượng sản phẩm khác nhau mà không cần biết chi tiết khởi tạo. Điều này giúp đơn giản hóa việc tạo đối tượng, đồng thời dễ mở rộng khi có thêm loại sản phẩm mới.

Mô tả các thành phần:

- ProductFactory (Factory Class)
- Product (Abstract Product)
- Laptop, Phone, Tablet, Headphone, Smartwatch (Concrete Product)
- ProductService (Client)

Lợi ích:

- Ẩn chi tiết khởi tạo: Client chỉ cần gọi một phương thức tạo, không cần biết class cụ thể.
- Dễ mở rộng: Khi thêm sản phẩm mới, chỉ cần mở rộng trong factory mà không ảnh hưởng phần còn lại.
- Tăng tính đóng gói: Logic khởi tạo được gom vào một nơi.

Hạn chế :

- Hiệu suất khi khởi tạo phức tạp: Nếu việc khởi tạo sản phẩm (như tải dữ liệu từ cơ sở dữ liệu hoặc thực hiện tính toán phức tạp) tốn nhiều tài nguyên, việc gọi Factory Method liên tục có thể ảnh hưởng đến hiệu suất. Có thể khắc phục bằng sử dụng caching hoặc khởi tạo lười (lazy initialization) để lưu trữ các sản phẩm đã tạo.

Tình huống ví dụ: Admin muốn thêm sản phẩm Laptop, Tablet, hoặc SmartWatch. Dựa trên productType, factory sẽ tạo đúng đối tượng tương ứng – không cần phân nhánh nhiều lần trong controller hay service.

1.4.4 Observer – Cập nhật số lượng giỏ hàng

Mẫu thiết kế Observer là một mẫu thiết kế thuộc nhóm Behavioral Patterns (mẫu hành vi), được sử dụng để thiết lập mối quan hệ một-nhiều giữa các đối tượng. Khi trạng thái của một đối tượng (chủ thể - subject) thay đổi, tất cả các đối tượng phụ thuộc (người quan sát - observers) sẽ được thông báo và tự động cập nhật. Trong trường hợp này, Observer được áp dụng để thông báo các thay đổi trong giỏ hàng (thêm, xóa, chỉnh sửa số lượng sản phẩm) đến các thành phần liên quan như giao diện số lượng sản phẩm trong giỏ hàng hoặc tính năng tính tổng giá.

Lý do phù hợp:

- **Tự động đồng bộ:** Khi giỏ hàng thay đổi (ví dụ: người dùng thêm sản phẩm hoặc chỉnh sửa số lượng), các thành phần phụ thuộc (như giao diện, tổng giá, trạng thái kho) cần được cập nhật ngay lập tức. Observer đảm bảo các thành phần này nhận thông báo mà không cần gắn chặt vào logic của giỏ hàng.
- **Tách biệt trách nhiệm:** Logic xử lý giỏ hàng không cần trực tiếp gọi các phương thức cập nhật của giao diện hay kho, giảm sự phụ thuộc giữa các thành phần.

- **Dễ mở rộng:** Khi có thêm thành phần mới cần theo dõi giỏ hàng (ví dụ: hệ thống gợi ý sản phẩm), chỉ cần đăng ký thành phần đó làm observer.

Mô tả các thành phần:

- CartSubject (Subject)
- CartObserver (Observer)
- CartUpdateSubject (ConcreteSubject)
- CartWebSocketNotifier (ConcreteObsever)

Lợi ích:

- **Tự động đồng bộ:** Giao diện giỏ hàng, tổng giá, hoặc trạng thái kho được cập nhật ngay khi số lượng thay đổi.
- **Tách biệt trách nhiệm:** Logic giỏ hàng không phải trực tiếp xử lý việc thông báo đến các thành phần khác.
- **Mở rộng dễ dàng:** Dễ thêm các thành phần mới cần theo dõi giỏ hàng (ví dụ: hệ thống gợi ý sản phẩm).

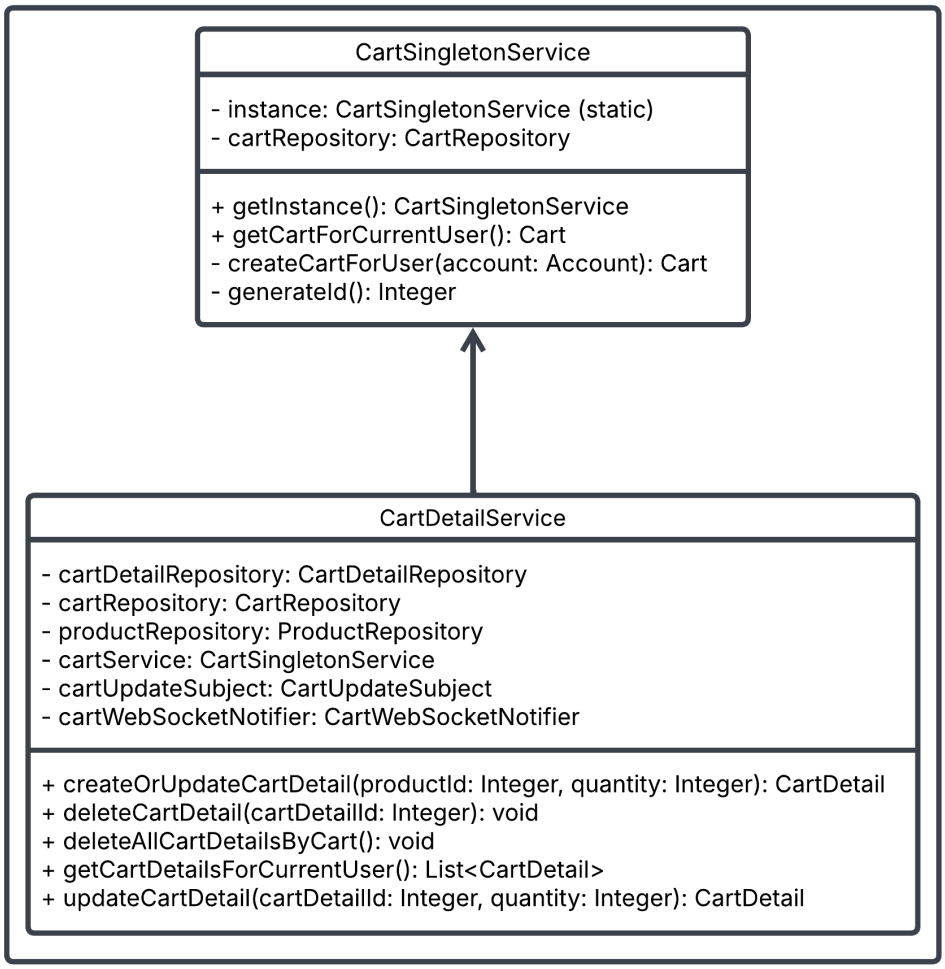
Hạn chế :

- **Phức tạp khi có nhiều đối tượng**: Khi có quá nhiều đối tượng quan sát (observers), việc quản lý và duy trì mối quan hệ giữa chúng có thể trở nên phức tạp và tốn kém tài nguyên.
- **Khó kiểm soát thứ tự thông báo**: Trong một số trường hợp, bạn có thể gặp phải vấn đề với thứ tự cập nhật, đặc biệt khi có nhiều observer được thông báo đồng thời.

Ví dụ: Khi người dùng tăng số lượng sản phẩm từ 1 lên 2, Observer thông báo đến giao diện để hiển thị số mới, đồng thời cập nhật tổng giá và kiểm tra kho hàng.

CHƯƠNG 2. LƯỢC ĐỒ CLASS THIẾT KẾ

2.1 Singleton Pattern



Hình 1: Singleton pattern

CartSingletonService (Subject - Singleton Class):

- Là lớp trung tâm triển khai mẫu thiết kế Singleton.
- Cung cấp một instance duy nhất để quản lý giỏ hàng của người dùng trong toàn bộ ứng dụng.
- Đảm bảo không tạo trùng giỏ hàng thông qua:
    - private static CartSingletonService instance
    - getInstance() – cung cấp quyền truy cập toàn cục

- Cung cấp phương thức như getInstance() để truy cập instance duy nhất và getCartForCurrentUser() để lấy giỏ hàng hiện tại.

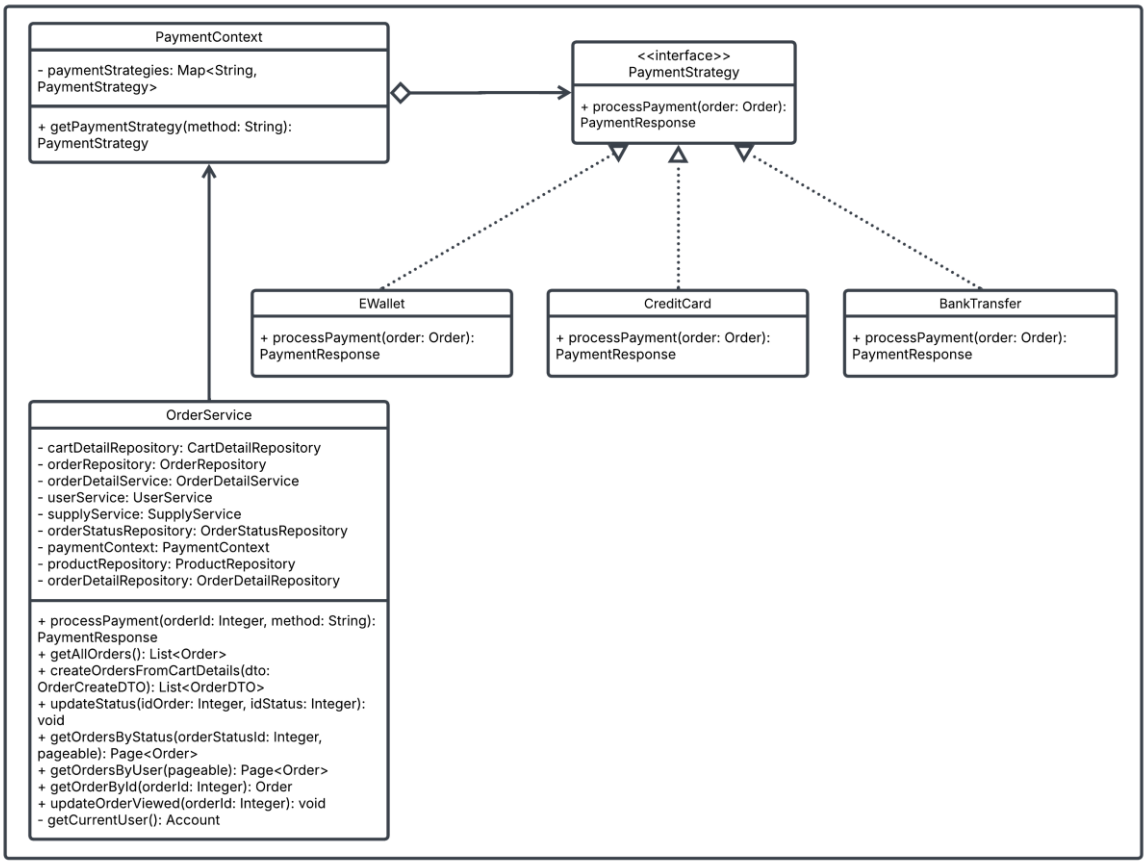
CartDetailService (Client):

- Là Client sử dụng Singleton để thao tác với giỏ hàng.
- Có trách nhiệm truy cập singleton object (CartSingletonService) thông qua getInstance() hoặc thông qua dependency injection (@Autowired).
- Ví dụ: CartDetailService gọi cartService.getCartForCurrentUser() để lấy giỏ hàng.

Mối quan hệ giữa các class:

- Mối quan hệ giữa CartSingletonService và CartDetailService là Association (Liên kết). Trong mối quan hệ này, CartDetailService sử dụng CartSingletonService để truy xuất giỏ hàng của người dùng thông qua phương thức getCartForCurrentUser(). CartDetailService không sở hữu hay kiểm soát vòng đời của CartSingletonService, mà chỉ tham chiếu đến nó để thực hiện các thao tác liên quan đến giỏ hàng như thêm, cập nhật, hoặc xóa sản phẩm. CartSingletonService, với tư cách là một Singleton, được quản lý bởi Spring Framework thông qua Dependency Injection (@Autowired), đảm bảo rằng chỉ có một thể hiện duy nhất của giỏ hàng cho mỗi người dùng trong suốt phiên làm việc. Mối quan hệ này mang tính chất tạm thời, vì CartDetailService chỉ tương tác với CartSingletonService khi cần thiết mà không lưu trữ tham chiếu lâu dài, giúp duy trì sự tách biệt giữa các lớp và dễ dàng mở rộng hệ thống.

2.2 Strategy Pattern



Hình 2: Strategy pattern

PaymentContext (Context):

- Là lớp quản lý và lựa chọn chiến lược thanh toán, sử dụng các strategy object (PaymentStrategy) để thực hiện thanh toán.
- Chỉ giao tiếp với các strategy object thông qua interface PaymentStrategy.
- Ví dụ: PaymentContext gọi strategy.processPayment(order) để xử lý thanh toán mà không cần biết loại thanh toán cụ thể (BankTransfer, CreditCard, v.v.).

PaymentStrategy (Strategy Interface):

- Là interface chung, định nghĩa phương thức processPayment(Order): PaymentResponse.
- Giúp các thuật toán thanh toán cụ thể có thể thay đổi linh hoạt tại runtime.
- Cho phép PaymentContext giao tiếp với các strategy object mà không phụ thuộc vào triển khai cụ thể.

BankTransfer, CreditCard, EWallet (Concrete Strategies):

- Là các lớp cụ thể triển khai interface PaymentStrategy.
- Mỗi lớp triển khai thuật toán thanh toán khác nhau:
    - BankTransfer: Xử lý thanh toán qua chuyển khoản ngân hàng.
    - CreditCard: Xử lý thanh toán qua thẻ tín dụng.
    - EWallet: Xử lý thanh toán qua ví điện tử.
- Triển khai phương thức processPayment() theo cách riêng.

OrderService (Client - Controller):

- Có trách nhiệm chọn và tạo strategy object dựa trên phương thức thanh toán (method).
- Truyền strategy object vào PaymentContext để sử dụng.
- Ví dụ: Trong OrderService.processPayment, gọi paymentContext.getPaymentStrategy(method) để chọn strategy và thực thi thanh toán.

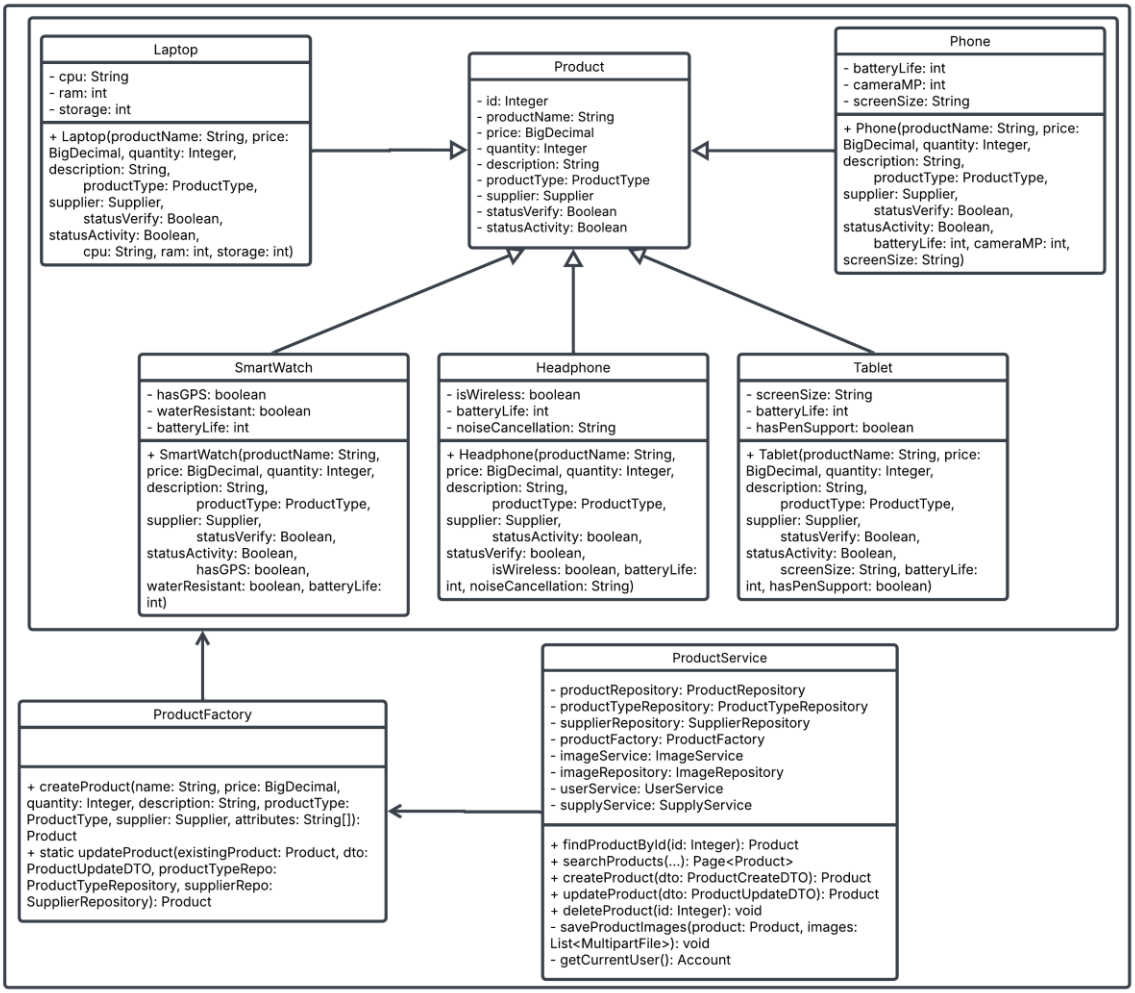
Mối quan hệ giữa các class:

- OrderService và PaymentContext: Association – OrderService sử dụng PaymentContext để lấy chiến lược thanh toán phù hợp dựa trên phương thức thanh toán người dùng yêu cầu, nhưng không sở hữu PaymentContext.
- PaymentContext và PaymentStrategy là mối quan hệ Aggregation vì PaymentContext chứa các PaymentStrategy nhưng không sở hữu chúng.

PaymentContext sử dụng một Map để lưu trữ các chiến lược thanh toán mà không tạo ra hay hủy bỏ các đối tượng PaymentStrategy.

- EWallet, BankTransfer, CreditCard và PaymentStrategy: Realization – EWallet, BankTransfer, CreditCard thực hiện các phương thức được định nghĩa trong PaymentStrategy.

2.3 Factory Pattern



Hình 3: Factory pattern

ProductService (Client):

- Là lớp sử dụng factory object (ProductFactory) để tạo hoặc cập nhật sản phẩm.
- Chỉ giao tiếp với factory object thông qua lớp trừu tượng ProductFactory.

- Lớp này sử dụng ProductFactory để tạo các đối tượng Product (Laptop, Phone, Tablet, v.v.). Lớp ProductService không biết chi tiết về cách tạo ra các sản phẩm, nó chỉ gọi phương thức createProduct() từ ProductFactory và truyền tham số để tạo sản phẩm.
- Ví dụ: ProductService gọi factory.createProduct(...) để tạo sản phẩm mà không cần biết loại sản phẩm cụ thể (Laptop, Phone, v.v.).

ProductFactory (Factory Class):

- Là interface hoặc lớp trừu tượng, cung cấp phương thức chung createProduct để tạo sản phẩm.
- Cho phép ProductService tạo sản phẩm mà không phụ thuộc vào triển khai cụ thể của factory.
- Ví dụ: ProductFactory định nghĩa createProduct(String, BigDecimal, Integer, String, ProductType, Supplier, String[]): Product.

LaptopFactory, PhoneFactory, TabletFactory, HeadphoneFactory, SmartwatchFactory (Concrete Product):

- Các lớp Laptop, Phone, Tablet, Headphone, và Smartwatch là các Concrete Products trong mô hình Factory Pattern**.**
- Mỗi lớp này là một sản phẩm cụ thể kế thừa từ Product và thực thi các chi tiết cụ thể cho từng loại sản phẩm.

Client (ProductService):

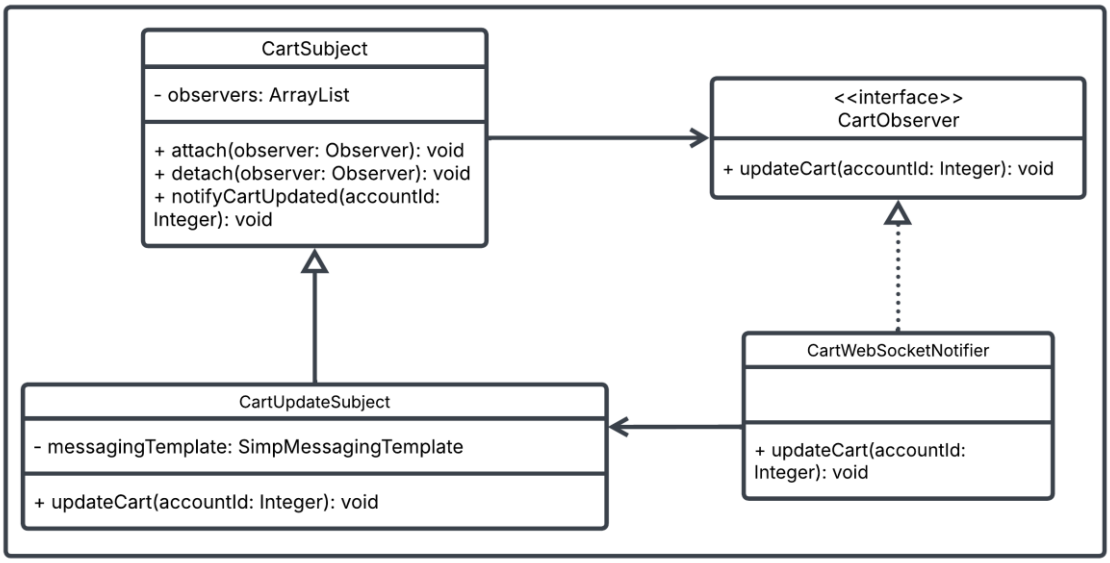
- Có trách nhiệm chọn và tạo factory object dựa trên ProductType.
- Truyền factory object vào ProductService để sử dụng (thực tế là gọi phương thức createProduct).
- ProductService là Client trong mô hình Factory Pattern. ProductService sẽ gọi ProductFactory.createProduct() để tạo sản phẩm mà không cần quan tâm đến

việc tạo ra sản phẩm như thế nào. ProductService chỉ yêu cầu sản phẩm từ Factory.

Mối quan hệ giữa các class:

- Mối quan hệ giữa Product và các lớp con như Laptop, Phone, Headphone, … thể hiện mối quan hệ Inheritance (Kế thừa)
- Mối quan hệ giữa ProductService và ProductFactory là Association vì ProductService sử dụng ProductFactory để tạo các đối tượng Product. ProductService tham chiếu đến ProductFactory để gọi phương thức createProduct(), nhưng không sở hữu hay quản lý ProductFactory.
- Mối quan hệ giữa ProductFactory và các Concrete Products là Association vì ProductFactory chỉ tạo ra các đối tượng cụ thể (như Laptop, Phone, Headphone) khi được yêu cầu, nhưng không sở hữu các đối tượng đó.

2.4 Observer Pattern



Hình 4: Factory pattern

CartSubject (Subject): Đối tượng được theo dõi, duy trì danh sách các Observer và thông báo khi trạng thái thay đổi.

CartObserver (Observer): Giao diện hoặc lớp trừu tượng định nghĩa phương thức mà các Observer cụ thể sẽ triển khai để nhận thông báo.

CartUpdateSubject (ConcreteSubject): Lớp cụ thể kế thừa hoặc triển khai Subject, chứa logic thực tế để quản lý trạng thái và thông báo.

CartWebSocketNotifier (ConcreteObsever): Lớp cụ thể triển khai giao diện Observer, định nghĩa hành động khi nhận thông báo.

Mối quan hệ giữa các class:

- Mối quan hệ giữa CartSubject và CartObserver là Association. Vì CartSubject có một danh sách các CartObserver, tức là CartSubject sẽ giữ tham chiếu đến các observer, nhưng không sở hữu chúng. Các observer sẽ được thông báo khi có sự thay đổi trong CartSubject. CartSubject không quản lý các observer, nó chỉ gửi thông báo khi có sự thay đổi.
- Mối quan hệ giữa CartSubject và CartUpdateSubject là Inheritance (Kế thừa) vì CartUpdateSubject là lớp kế thừa từ CartSubject, mở rộng và bổ sung chức năng notifyCartUpdated() để gửi thông báo giỏ hàng đã thay đổi thông qua WebSocket.
- Mối quan hệ giữa CartUpdateSubject và CartWebSocketNotifier là Association. CartUpdateSubject có một hoặc nhiều CartObserver, trong đó CartWebSocketNotifier là một observer. Khi CartUpdateSubject gửi thông báo (thông qua phương thức notifyCartUpdated()), CartWebSocketNotifier sẽ nhận và xử lý thông báo đó.

CHƯƠNG 3. HIỆN THỰC MẪU

3.1 Singleton Pattern

Singleton Pattern được sử dụng để đảm bảo rằng mỗi người dùng chỉ có một giỏ hàng duy nhất trong suốt phiên làm việc.

Dữ liệu giỏ hàng (product list, quantity,...) sẽ được quản lý tập trung trong một thể hiện singleton duy nhất cho mỗi người dùng.

Mô tả luồng hoạt động (phía server):

- Khi người dùng thực hiện các thao tác với giỏ hàng như thêm sản phẩm, cập nhật số lượng, xóa sản phẩm… từ bất kỳ vị trí nào trong hệ thống: Các thành phần sẽ gọi cartService.getCartForCurrentUser().
- Phương thức này kiểm tra xem đã tồn tại giỏ hàng cho người dùng chưa: Nếu chưa có, CartSingletonService sẽ khởi tạo giỏ hàng mới và lưu vào cơ sở dữ liệu. Nếu đã tồn tại, nó sẽ trả về giỏ hàng hiện tại.
- Mọi thao tác (add, update, remove) sẽ đều tác động lên cùng một thể hiện Cart duy nhất.

Hoạt động từ phía người dùng:

- Người dùng đăng nhập, duyệt sản phẩm và thêm sản phẩm vào giỏ hàng từ nhiều trang khác nhau: trang chi tiết, popup khuyến mãi, trang thanh toán…
- Nhờ Singleton, toàn bộ sản phẩm được gom lại trong một giỏ hàng duy nhất.
- Khi chuyển sang thanh toán, hệ thống hiển thị đúng nội dung người dùng đã chọn từ trước.

3.2 Strategy Pattern

Strategy Pattern được dùng để linh hoạt xử lý nhiều phương thức thanh toán khác nhau như: Chuyển khoản, Ví điện tử, Thẻ tín dụng.

Mô tả luồng hoạt động (phía code/server):

- Khi người dùng chọn một phương thức thanh toán và gửi yêu cầu thanh toán, Controller gọi đến PaymentContext.getPaymentStrategy(method) để lấy chiến lược phù hợp.
- Chiến lược tương ứng (Concrete Strategy) được chọn sẽ xử lý thanh toán bằng cách cài đặt cụ thể cho processPayment(order). Gồm: cập nhật trạng thái đơn hàng → gán phương thức thanh toán → trả về PaymentResponse.
- Mỗi Strategy sẽ cập nhật trạng thái đơn hàng, phương thức thanh toán, và trả về PaymentResponse.

Hoạt động từ phía người dùng:

- Người dùng đặt hàng và chuyển sang bước thanh toán.
- Họ chọn "Chuyển khoản ngân hàng" và nhấn "Xác nhận".
- Hệ thống xử lý theo chiến lược BankTransfer: cập nhật đơn hàng là đã thanh toán, gửi thông báo thành công.
- Nếu chọn phương thức khác như "Ví điện tử", hệ thống chỉ cần thay thế bằng strategy tương ứng mà không đổi logic chính.

3.3 Factory Pattern

Factory Pattern được dùng để tạo ra đối tượng sản phẩm phù hợp với từng loại như: Laptop, Phone, Tablet, SmartWatch...

Dựa trên thuộc tính ProductType, factory tạo đúng instance tương ứng (subclass của Product).

Mô tả luồng hoạt động (phía code/server):

- Khi Client (thường là Controller hoặc Service) gửi yêu cầu tạo sản phẩm, Controller nhận yêu cầu và chuyển đến ProductService.

- ProductService gọi ProductFactory.createProduct(...), truyền vào các thuộc tính cần thiết và tên loại sản phẩm.
- Factory kiểm tra productTypeName và tạo instance tương ứng (new Laptop(...), new Phone(...),...).
- Instance được trả về và lưu vào database thông qua ProductRepository.

Hoạt động từ phía người dùng:

- Admin truy cập trang quản lý sản phẩm.
- Chọn loại sản phẩm (ví dụ: Laptop) và nhập các thông tin chi tiết (RAM, CPU, bộ nhớ...).
- Nhấn nút “Lưu”.
- Hệ thống backend sử dụng ProductFactory để tạo đúng class Laptop.
- Sản phẩm được lưu trữ, hiển thị đúng thông tin trên hệ thống.
- Nếu sau này thêm loại sản phẩm mới (ví dụ: Camera), chỉ cần tạo subclass mới và mở rộng trong ProductFactory mà không cần sửa các phần khác trong hệ thống.

3.4 Observer Pattern

Trong hệ thống, Observer Pattern được sử dụng để theo dõi và phản hồi các thay đổi trong giỏ hàng (cart) của người dùng. Mô hình gồm hai phần chính:

- CartSubject (Subject): quản lý danh sách các CartObserver – những thành phần muốn được thông báo khi giỏ hàng thay đổi.
- CartObserver (Observer): định nghĩa giao diện chung để các lớp lắng nghe và xử lý khi có sự thay đổi.

Luồng hoạt động trên phía code (server):

- Khi có sự kiện thay đổi giỏ hàng (thêm, xóa sản phẩm…), đối tượng CartSubject sẽ gọi notifyCartUpdated(accountId).
- Phương thức này lần lượt gọi updateCart(accountId) trên tất cả các observer đã đăng ký.
- Một trong các observer là CartWebSocketNotifier – nơi xử lý việc gửi thông báo qua WebSocket đến client.
- Ngoài ra, lớp CartUpdateSubject là một subclass của CartSubject, có thể dùng thêm SimpMessagingTemplate để gửi thông báo đến từng WebSocket topic cụ thể (theo accountId).
- Kết quả: bất kỳ client nào đã đăng ký theo dõi /topic/cart/{accountId} đều sẽ nhận được thông báo "Cart updated" ngay khi có thay đổi.

Hoạt động từ phía người dùng:

- Người dùng đăng nhập và mở giao diện giỏ hàng (Cart page).
- Khi người dùng thêm sản phẩm vào giỏ hàng hoặc một người khác (Admin, thiết bị khác) cập nhật giỏ hàng của họ: Trên server: hệ thống gọi notifyCartUpdated(accountId). Server gửi thông báo WebSocket tới topic /topic/cart/{accountId}.
- Ở phía client (trình duyệt), nếu người dùng đang kết nối WebSocket: Họ sẽ nhận được thông báo real-time mà không cần tải lại trang. Giao diện có thể hiển thị pop-up, badge số lượng cập nhật, hoặc tự động đồng bộ giỏ hàng.

CHƯƠNG 4. XỬ LÝ DỮ LIỆU

4.1 Lưu trữ dữ liệu – Repository pattern

4.1.1 Phân tích mẫu thiết kế

Mẫu Repository Pattern là một mẫu thiết kế thuộc nhóm Architectural Patterns, được sử dụng để tách biệt logic nghiệp vụ khỏi logic truy cập dữ liệu. Repository cung cấp một giao diện thống nhất (như save(), findById(), delete()) để thao tác với dữ liệu, bất kể nguồn dữ liệu là cơ sở dữ liệu, file, hay API.

Lý do phù hợp :

- Tách biệt logic: Repository giúp logic nghiệp vụ không phụ thuộc vào cách dữ liệu được lưu trữ (SQL, NoSQL, file, v.v.), làm cho mã nguồn rõ ràng và dễ bảo trì.
- Chuẩn hóa giao diện: Cung cấp các phương thức chung để thao tác dữ liệu, giúp ứng dụng tương tác với dữ liệu một cách nhất quán.
- Dễ mở rộng: Khi cần chuyển sang nguồn dữ liệu mới (ví dụ: từ MySQL sang MongoDB), chỉ cần tạo một triển khai repository mới mà không cần sửa logic nghiệp vụ

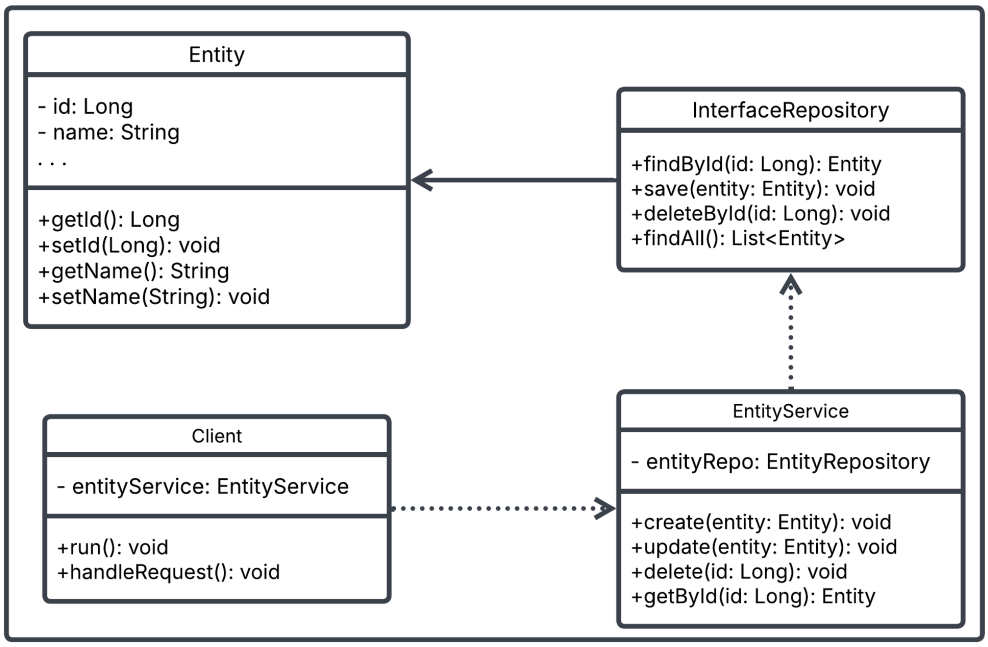
Lợi ích

- Tính trừu tượng cao: Repository che giấu chi tiết triển khai của tầng lưu trữ dữ liệu, giúp logic nghiệp vụ chỉ làm việc với các phương thức trừu tượng như save(), findById(). Điều này làm giảm sự phụ thuộc vào công nghệ cụ thể.
- Tái sử dụng mã: Các phương thức truy vấn hoặc lưu trữ dữ liệu được định nghĩa một lần trong repository và có thể được sử dụng ở nhiều nơi trong ứng dụng.
- Dễ bảo trì và mở rộng: Khi cần thêm chức năng mới (ví dụ: lưu dữ liệu vào một nguồn mới), chỉ cần tạo một triển khai repository mới mà không cần sửa đổi logic nghiệp vụ.

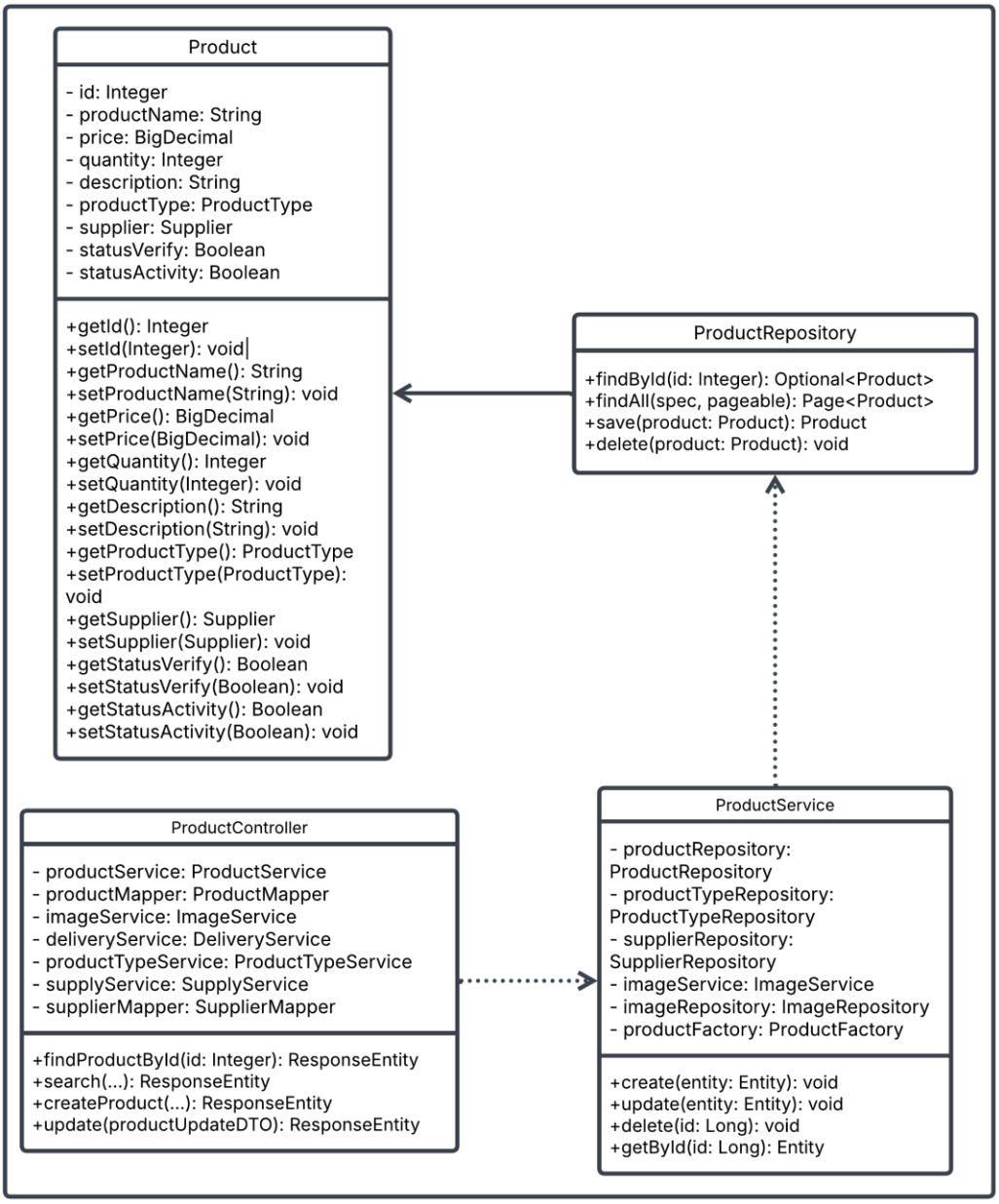
- Tăng tính kiểm thử: Repository cho phép thay thế dễ dàng bằng các mock object trong unit test, giúp kiểm tra logic mà không cần kết nối đến cơ sở dữ liệu thật.
- Nhất quán: Repository đảm bảo rằng mọi thao tác dữ liệu đều tuân theo một giao diện chung, giảm nguy cơ sai sót hoặc không đồng bộ.

Ví dụ: Một ứng dụng quản lý hóa đơn cần lưu trữ và truy xuất thông tin hóa đơn từ cơ sở dữ liệu. Chúng ta sẽ sử dụng Repository Pattern để xử lý việc lưu trữ dữ liệu.

4.1.2 Lược đồ class thiết kế



Hình 5: Repository pattern



Hình 6: Repository pattern – Product

- Entity (Model): Entity là một lớp mô hình (model) đại diện cho một bản ghi trong cơ sở dữ liệu hoặc một đối tượng trong hệ thống. Nó chứa các thuộc tính (fields) tương ứng với cột trong bảng dữ liệu và các phương thức getter/setter để truy cập. Entity không chứa logic nghiệp vụ, chỉ đơn thuần là cấu trúc dữ liệu.
- InterfaceRepository: định nghĩa một tập hợp các phương thức để thao tác với dữ liệu (CRUD). Nó cung cấp một giao diện trừu tượng, che giấu chi tiết triển

khai của nguồn dữ liệu (ví dụ: cơ sở dữ liệu SQL, NoSQL, file). Đảm bảo logic nghiệp vụ không phụ thuộc vào cách dữ liệu được lưu trữ.

- EntityService: Service Layer chứa logic nghiệp vụ của ứng dụng, ví dụ: kiểm tra dữ liệu, thực hiện các quy tắc kinh doanh, hoặc phối hợp nhiều thao tác dữ liệu. Nó gọi đến Repository để thực hiện các thao tác lưu trữ, nhưng không trực tiếp làm việc với cơ sở dữ liệu. Service Layer giúp tách biệt logic nghiệp vụ khỏi tầng dữ liệu và tầng giao tiếp với client
- Client: Client là tầng giao tiếp với người dùng hoặc hệ thống bên ngoài (ví dụ: giao diện người dùng, API, hoặc ứng dụng khác). Nó gửi yêu cầu đến Service Layer và nhận kết quả để hiển thị hoặc xử lý tiếp. Client không biết đến repository hay cơ sở dữ liệu, chỉ làm việc với service.
- Để minh họa cho việc áp dụng Repository Pattern trong hệ thống, em đã xây dựng hai sơ đồ class cụ thể:
  - *Hình 5* là sơ đồ class tổng quát của Repository Pattern, thể hiện rõ vai trò và mối quan hệ giữa các thành phần chính gồm: Entity, Repository Interface, Service Layer, và Client. Trong mô hình này, Entity là lớp ánh xạ dữ liệu, chỉ chứa các thuộc tính tương ứng với cột trong cơ sở dữ liệu và các phương thức getter/setter đơn giản. Repository Interface định nghĩa các phương thức thao tác với dữ liệu (như findById, save, delete...), giúp che giấu chi tiết lưu trữ thực tế và cung cấp giao diện trừu tượng cho Service sử dụng. Service xử lý logic nghiệp vụ, thực hiện các bước kiểm tra, tính toán và gọi Repository để lưu hoặc truy vấn dữ liệu. Client (ví dụ: Controller hoặc lớp giao diện) là tầng tiếp xúc với người dùng, gửi yêu cầu đến Service và nhận kết quả phản hồi mà không tương tác trực tiếp với Repository hay Entity.
  - *Hình 6* là sơ đồ class cụ thể cho thực thể Product trong hệ thống thương mại điện tử. Sơ đồ này mô tả rõ cấu trúc của lớp Product – một entity tiêu biểu, với các thuộc tính như id, productName, price, quantity, description,

cùng với mối quan hệ đến các entity khác như ProductType và Supplier thông qua annotation @ManyToOne. Ngoài ra, lớp còn bao gồm các phương thức getter và setter để truy cập các thuộc tính. Sơ đồ này giúp thể hiện cách một thực thể dữ liệu cụ thể được định nghĩa và sử dụng trong toàn bộ luồng xử lý của Repository Pattern.

- Lớp Product trong *Hình 6* được chọn làm ví dụ điển hình để minh họa cho cách áp dụng Repository Pattern trong một thực thể cụ thể. Tuy nhiên, trong toàn bộ hệ thống, đây chỉ là một phần nhỏ trong cấu trúc dữ liệu tổng thể. Chương trình còn bao gồm nhiều entity quan trọng khác như: Cart, Order, Supply, User, Invoice, v.v... Mỗi đối tượng này đều được thiết kế theo nguyên tắc tương tự với Product, nghĩa là đều có lớp Entity riêng, Repository để truy cập dữ liệu, Service để xử lý nghiệp vụ và được điều phối bởi các lớp Controller. Bên cạnh các entity chính, hệ thống còn bao gồm nhiều thực thể phụ trợ như ProductType, Review, Image, Delivery, Supplier,... nhằm hỗ trợ mở rộng dữ liệu, tổ chức logic nghiệp vụ theo hướng module hóa và đảm bảo khả năng mở rộng cho hệ thống. Tất cả những thành phần này đều được thiết kế và kết nối thống nhất thông qua mô hình Repository Pattern, đảm bảo tính tách biệt giữa các tầng, tính tái sử dụng và dễ dàng bảo trì trong tương lai.

Mối quan hệ giữa các class:

- Mối quan hệ giữa Entity và InterfaceRepository là Association, vì InterfaceRepository là nơi chứa các phương thức để thao tác với Entity. InterfaceRepository sử dụng các đối tượng Entity trong các phương thức của nó, ví dụ như findById(), save(), delete(). Nhưng InterfaceRepository không sở hữu Entity, nó chỉ thực hiện thao tác với các đối tượng Entity.
- EntityService phụ thuộc vào InterfaceRepository để thực hiện các thao tác CRUD đối với Entity. EntityService gọi các phương thức của InterfaceRepository để truy xuất dữ liệu từ cơ sở dữ liệu.

- Mối quan hệ giữa Client(Controller) và EntityService sẽ là Dependency. Vì Controller chỉ là lớp tiếp nhận và xử lý các yêu cầu HTTP, không thực hiện nghiệp vụ phức tạp. Nó chỉ cần gọi Service để xử lý nghiệp vụ. Service thực hiện các nghiệp vụ logic và không cần biết về HTTP hoặc các yêu cầu từ người dùng. Nó chỉ cung cấp các phương thức phục vụ cho Controller.

4.1.3 Hiện thực mẫu

Repository Pattern được sử dụng để tách biệt logic truy cập dữ liệu ra khỏi logic nghiệp vụ trong hệ thống. Nó giúp đảm bảo rằng tầng nghiệp vụ (service) không phụ thuộc trực tiếp vào cách dữ liệu được lưu trữ (database, file, API...).

Luồng hoạt động trong hệ thống (phía code):

- Người dùng gửi yêu cầu (ví dụ: xem sản phẩm, thêm đơn hàng...) thông qua giao diện người dùng (gọi API từ trình duyệt, mobile app...).
- Controller nhận yêu cầu và chuyển đến lớp Service để xử lý nghiệp vụ.
- Service thực hiện các logic liên quan như kiểm tra dữ liệu, xử lý liên kết với các thực thể khác.
- Khi cần truy cập cơ sở dữ liệu, Service gọi đến Repository: Repository ở đây là interface kế thừa từ JpaRepository hoặc CrudRepository, cung cấp các phương thức như findById, save, deleteById, findAll,... Spring Data sẽ tự động sinh ra phần cài đặt (không cần viết RepositoryImpl).
- Kết quả từ repository được trả về lại cho Service, sau đó Controller, và cuối cùng là phản hồi cho người dùng.

Hoạt động từ phía người dùng

- Người dùng mở trang chi tiết sản phẩm của sản phẩm có ID là 101.
- Trình duyệt gửi yêu cầu HTTP GET đến endpoint /api/products/101.
- ProductController gọi ProductService.findProductById(101).

- ProductService gọi ProductRepository.findById(101).
- Repository truy vấn database và trả về một đối tượng Product.
- Dữ liệu sản phẩm được trả về client dưới dạng JSON và hiển thị trên giao diện

4.2 Xuất dữ liệu – Strategy pattern

4.2.1 Phân tích mẫu thiết kế

Mẫu Strategy Pattern là một mẫu thiết kế thuộc nhóm Behavioral Patterns, được sử dụng để định nghĩa một tập hợp các thuật toán (chiến lược) có thể hoán đổi lẫn nhau. Trong trường hợp này, Strategy Pattern được áp dụng để xử lý việc xuất hóa đơn dưới các định dạng khác nhau như CSV, JSON, PDF

Lý do phù hợp vì Strategy Pattern xử lý yêu cầu xuất hóa đơn với các định dạng khác nhau như CSV, JSON, PDF. Đây là lựa chọn phù hợp khi các định dạng xuất dữ liệu có thể thay đổi độc lập và cần được hoán đổi linh hoạt tại thời điểm chạy. Strategy Pattern cho phép định nghĩa một giao diện xuất dữ liệu chung, trong khi từng định dạng cụ thể sẽ được đóng gói thành các lớp riêng biệt và dễ dàng cắm vào hệ thống mà không ảnh hưởng đến logic tổng thể.

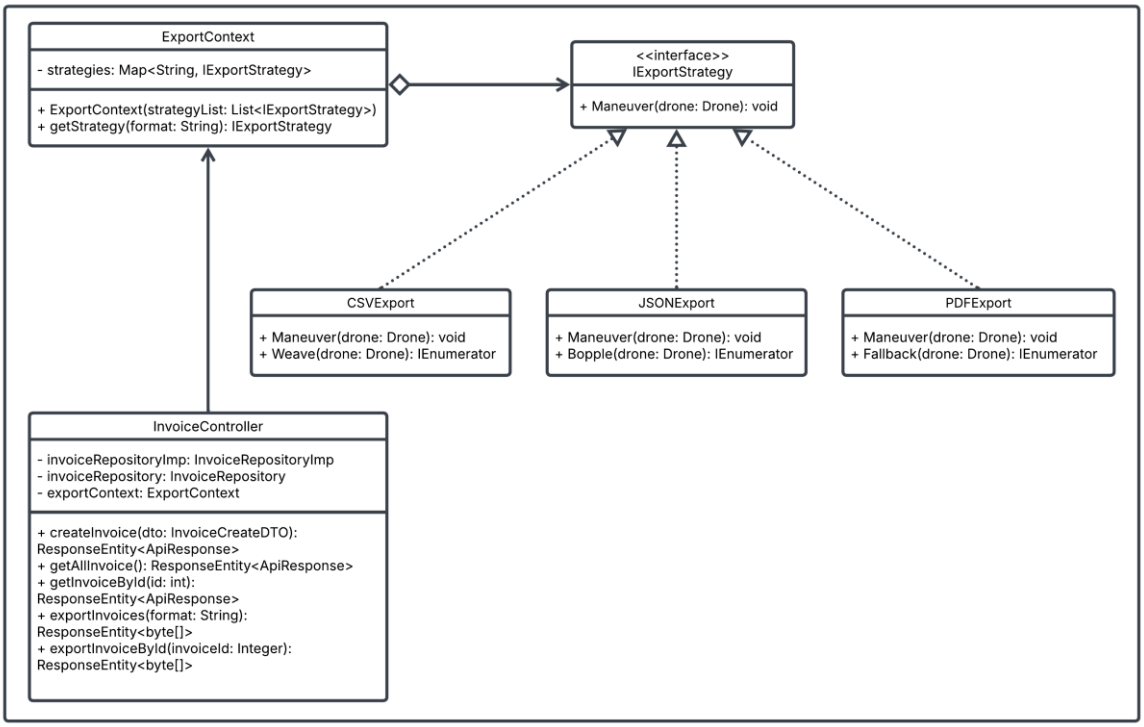
Lợi ích:

- Linh hoạt và mở rộng tốt: Mỗi định dạng (CSV, JSON, PDF...) được triển khai trong một lớp riêng (CSVExport, JSONExport, PDFExport), và có thể dễ dàng thêm định dạng mới mà không cần sửa đổi logic hiện có.
- Tuân thủ nguyên tắc Open/Closed: Các chiến lược export tuân theo giao diện IExportStrategy, giúp hệ thống mở rộng nhưng không sửa đổi code cũ.
- Tách biệt rõ ràng trách nhiệm: Lớp ExportContext đóng vai trò điều phối và lựa chọn chiến lược phù hợp, đảm bảo Controller không cần biết chi tiết các định dạng.

Ví dụ: Khi người dùng yêu cầu xuất hóa đơn dưới dạng PDF, ExportContext sẽ chọn PDFExport – một lớp triển khai IExportStrategy. Quá trình xuất dữ liệu

sẽ được thực hiện thông qua chiến lược phù hợp, đảm bảo tính đóng gói, mở rộng và dễ bảo trì. Việc chuyển đổi sang xuất CSV hoặc JSON chỉ đơn giản là thay đổi định dạng truyền vào context mà không cần sửa đổi bất kỳ dòng code nào trong controller hoặc service.

4.2.2 Lược đồ class thiết kế



Hình 7: Strategy pattern export

- *ExportContext (Context):* điều phối, cung cấp chiến lược phù hợp cho Client. Giữ và quản lý danh sách các IExportStrategy.Chọn đúng chiến lược (Concrete Strategy) tương ứng theo định dạng đầu vào (csv, json, pdf)
- *IExportStrategy (Strategy):* Định nghĩa giao diện chung cho các cách export hóa đơn khác nhau. Cho phép mở rộng nhiều định dạng export khác nhau mà không cần thay đổi logic phía Client hoặc Context.
- *CSVExport, JSONExport, PDFExport (Concrete Strategy):* Implement để export các định dạng khác nhau

- *InvoiceController (Client):* là lớp tiếp nhận yêu cầu từ người dùng, xử lý các thao tác liên quan đến hóa đơn như tạo mới, lấy danh sách, xem chi tiết và gửi yêu cầu xuất hóa đơn với định dạng cụ thể (CSV, PDF, JSON...) thông qua ExportContext

Mối quan hệ giữa các class:

- Mối quan hệ giữa ExportContext và IexportStrategy là Aggregation. Vì ExportContext chứa một danh sách các IExportStrategy (CSV, PDF, v.v.). ExportContext sử dụng các chiến lược (các đối tượng IExportStrategy), nhưng không quản lý vòng đời của chúng. Các chiến lược có thể được tạo và sử dụng độc lập, và ExportContext chỉ là lớp quản lý các chiến lược.

- Mối quan hệ giữa CSVExport, PDFExport, JSONExport và IexportStrategy là Realization. Vì CSVExport triển khai tất cả các phương thức trong IExportStrategy và cung cấp hành vi cụ thể cho phương thức export(), getContentType(), và getFileName().

- Mối quan hệ giữa InvoiceController và ExportContext là Association, vì InvoiceController chỉ tham chiếu đến ExportContext để lấy chiến lược xuất mà không sở hữu hoặc quản lý ExportContext.

4.2.3 Thực hiện mẫu

Tính năng xuất hóa đơn được xây dựng dựa trên Strategy Pattern, cho phép hệ thống dễ dàng hỗ trợ nhiều định dạng đầu ra khác nhau như CSV, JSON, và PDF.

Luồng hoạt động trong hệ thống (phía code):

- Người dùng yêu cầu xuất hóa đơn theo định dạng cụ thể (ví dụ: "pdf" hoặc "csv").

- Controller tiếp nhận yêu cầu và gọi đến ExportContext.

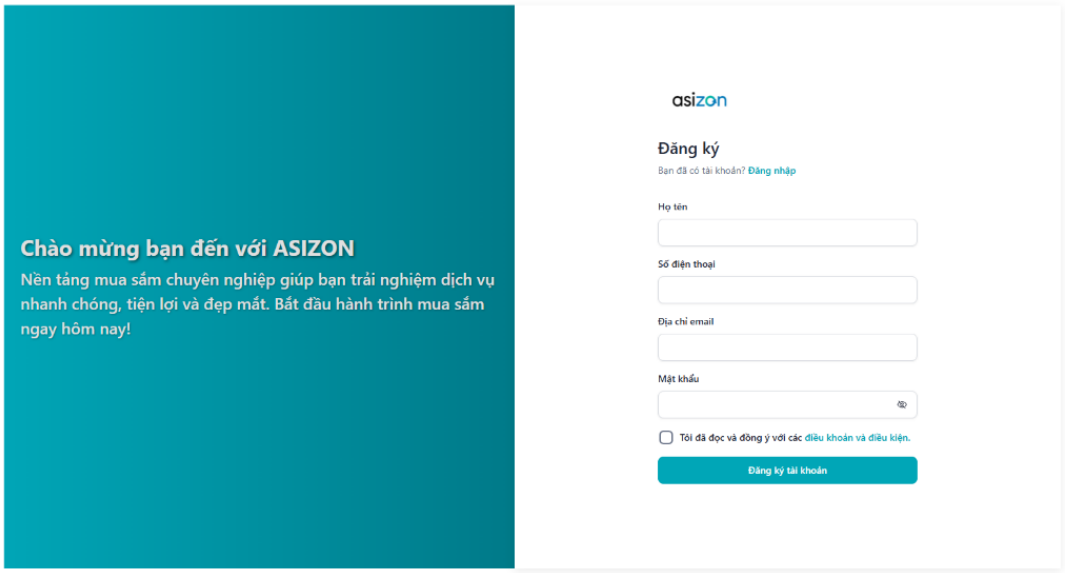
- ExportContext đóng vai trò là Context của Strategy Pattern: Nó sẽ lựa chọn và trả về một đối tượng IExportStrategy phù hợp (ví dụ PDFExport, CSVExport, JSONExport) dựa trên định dạng đầu vào.
- Sau đó, phương thức export(List<Invoice>) của chiến lược cụ thể sẽ được gọi để: Lấy dữ liệu hóa đơn, định dạng nội dung theo kiểu mong muốn, trả về mảng byte[] để tải file.
- Controller sử dụng thông tin từ strategy (MIME type, tên file) để tạo phản hồi HTTP trả về file cho client.

Hoạt động từ phía người dùng

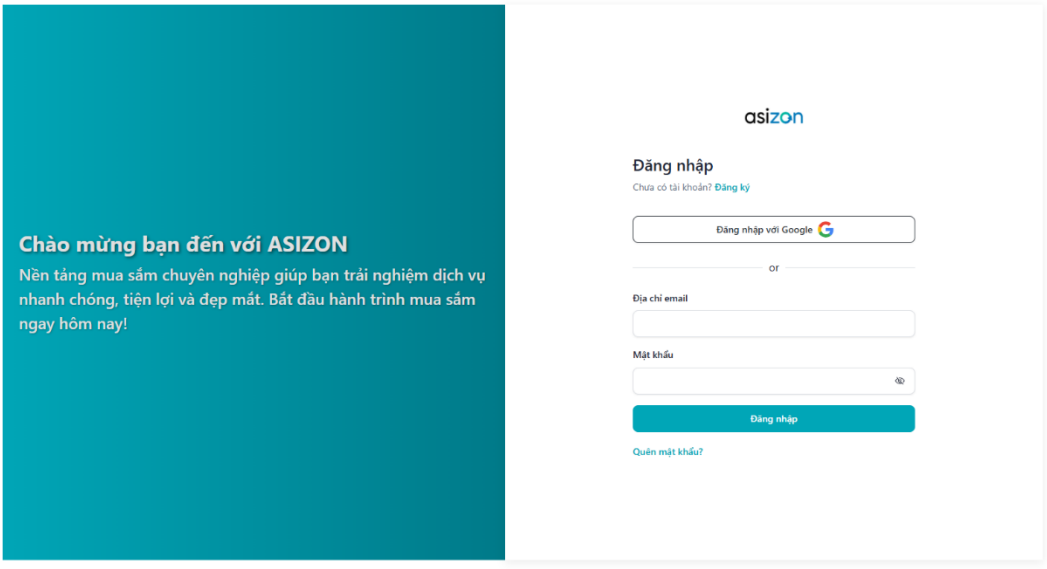
- Người dùng truy cập trang quản lý hóa đơn trên giao diện web.
- Họ chọn định dạng cần tải (CSV / PDF / JSON) từ dropdown và nhấn nút "Xuất hóa đơn".
- Hệ thống gửi yêu cầu đến server kèm tham số định dạng (ví dụ: ?format=pdf).
- Sau vài giây, trình duyệt tự động tải về file invoices.pdf chứa nội dung hóa đơn được định dạng đầy đủ.
- Nếu chọn CSV, hệ thống sẽ trả về file .csv có thể mở bằng Excel. Nếu chọn JSON, hệ thống trả về file .json dạng máy đọc được (machine-readable).

CHƯƠNG 5. GIAO DIỆN

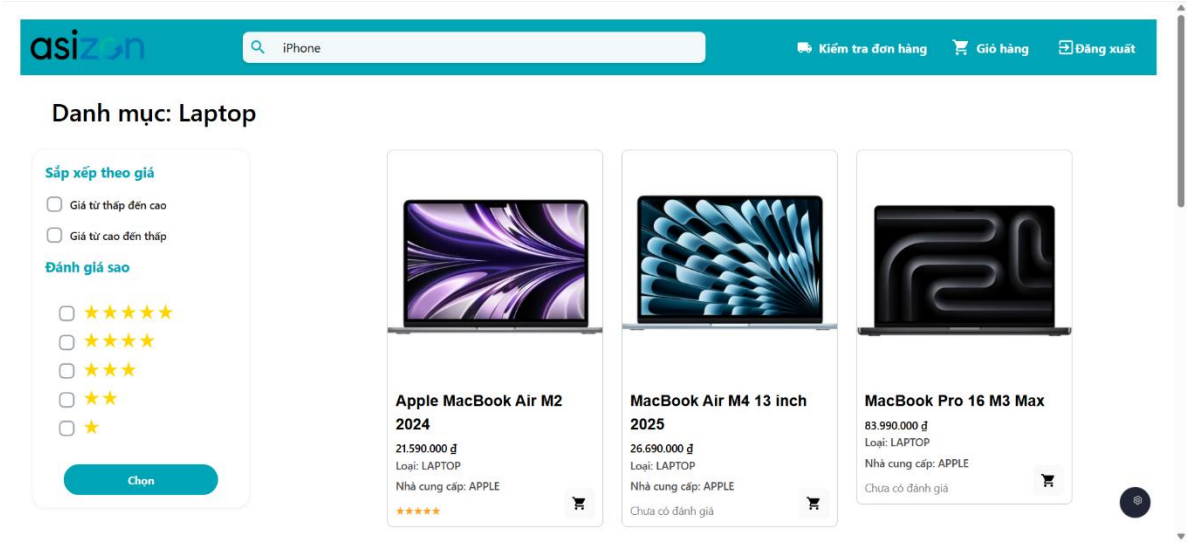
5.1 Giao diện từ người dùng



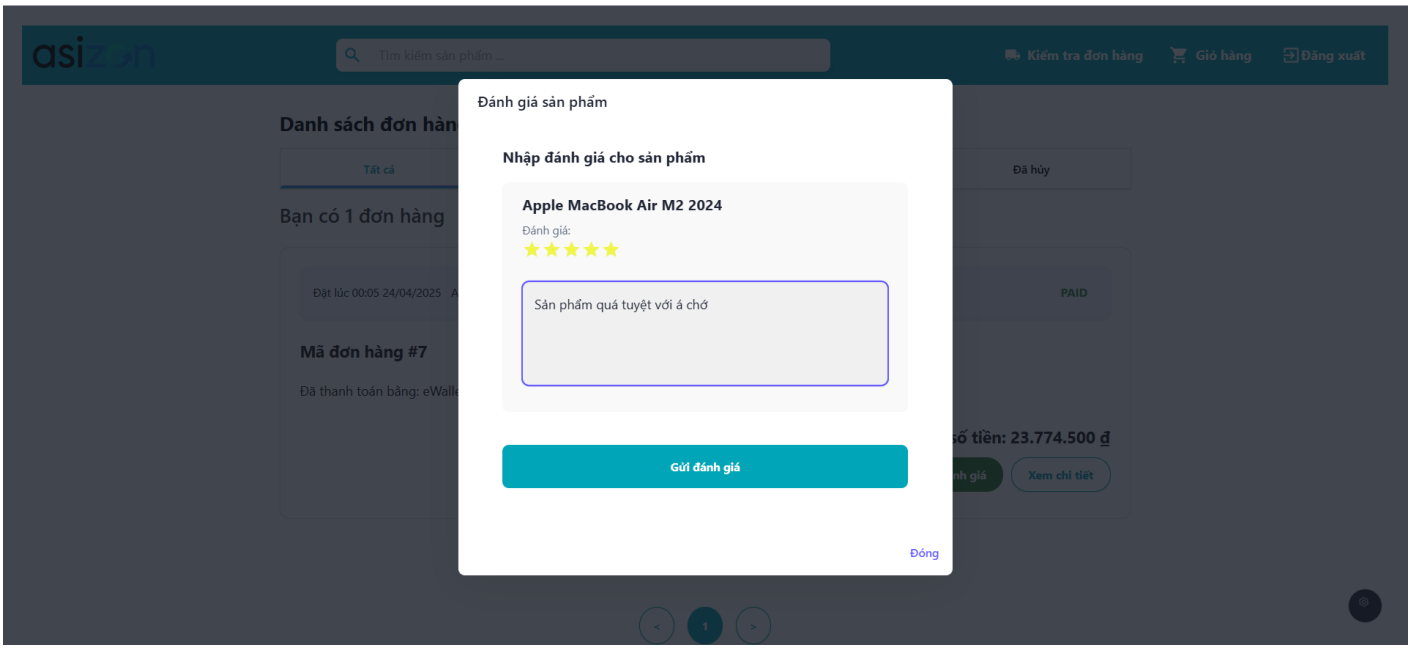
Hình 8: Trang đăng ký



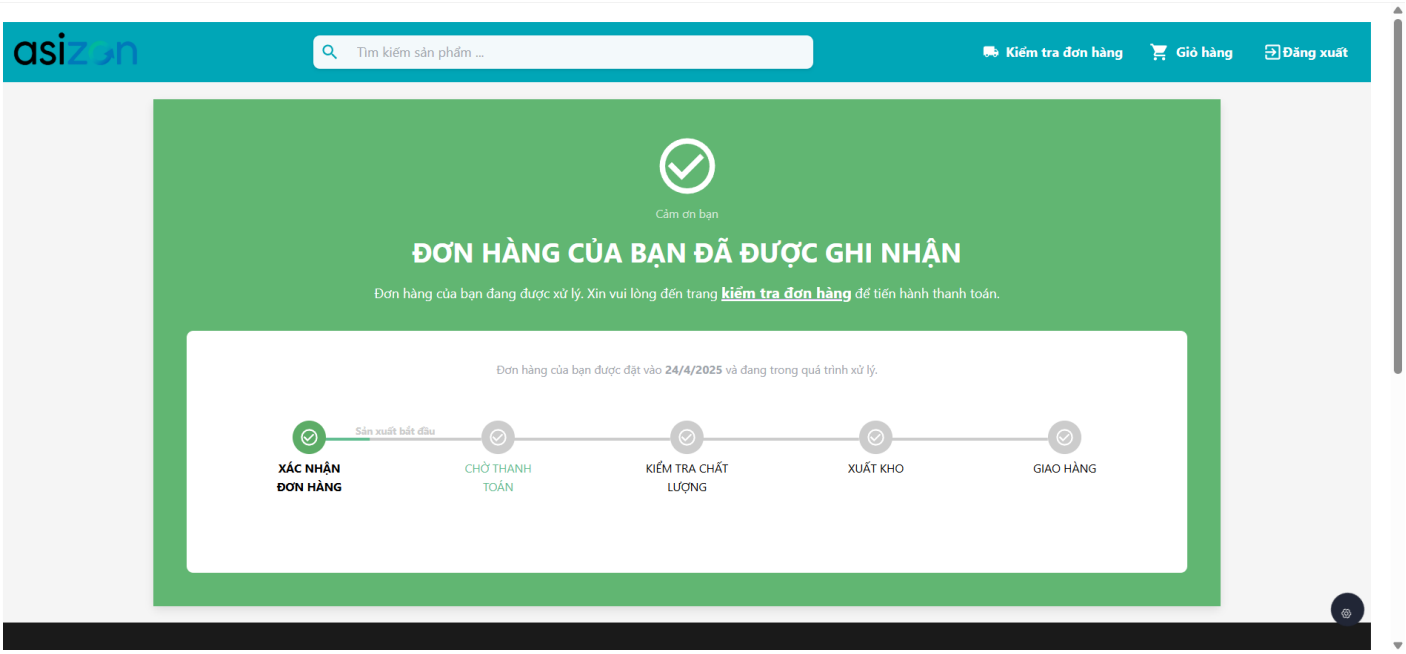
Hình 9: Trang đăng nhập



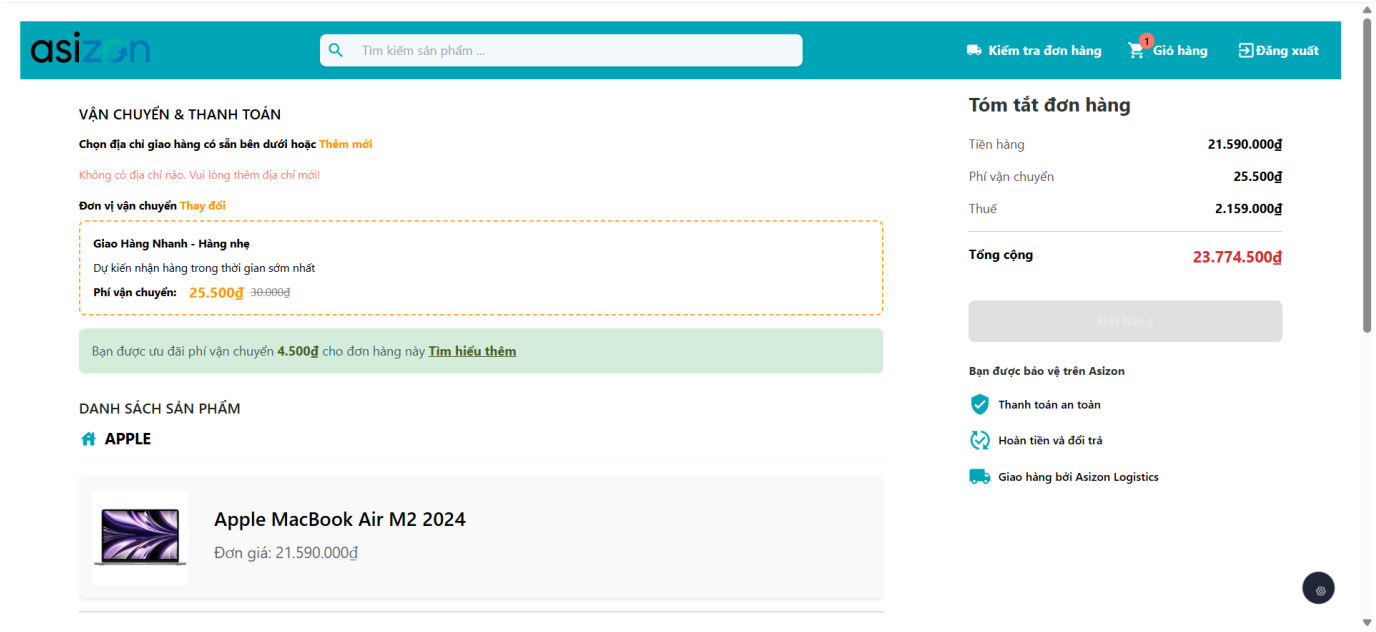
Hình 10: Danh mục sản phẩm



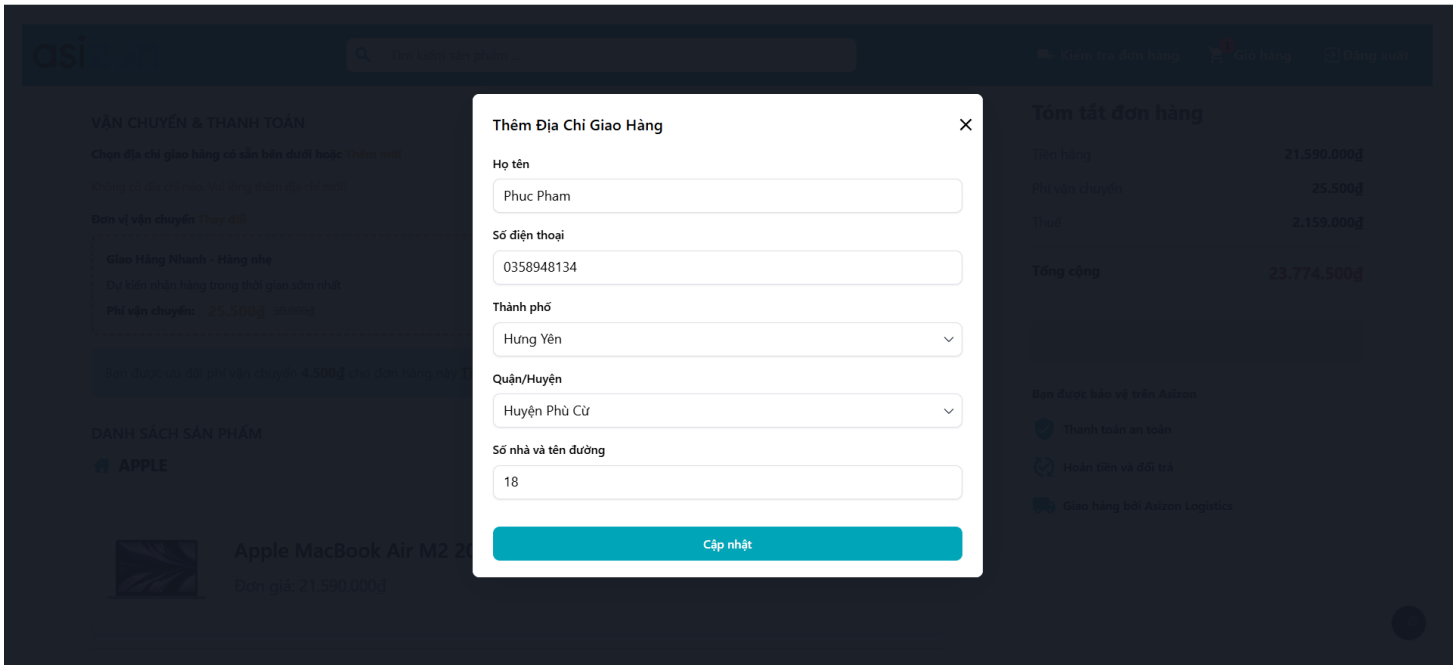
Hình 11: Đánh giá sản phẩm



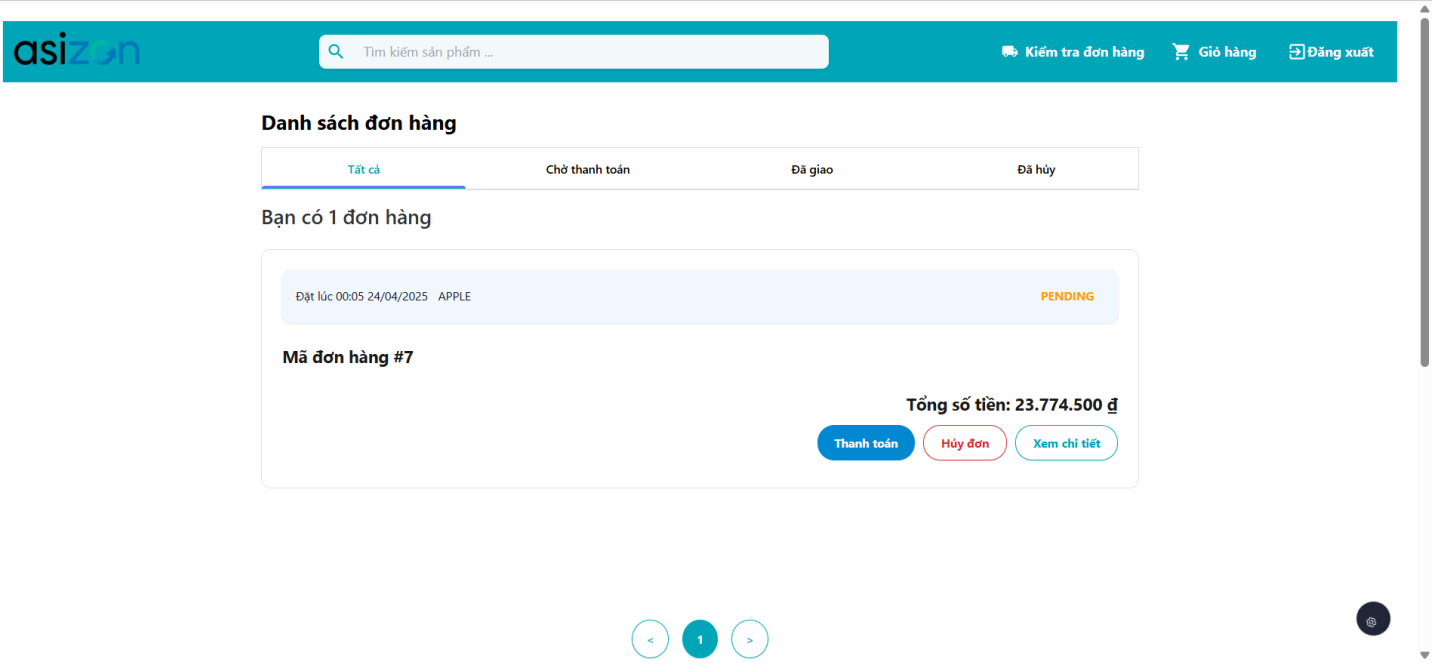
Hình 12: Đặt hàng thành công



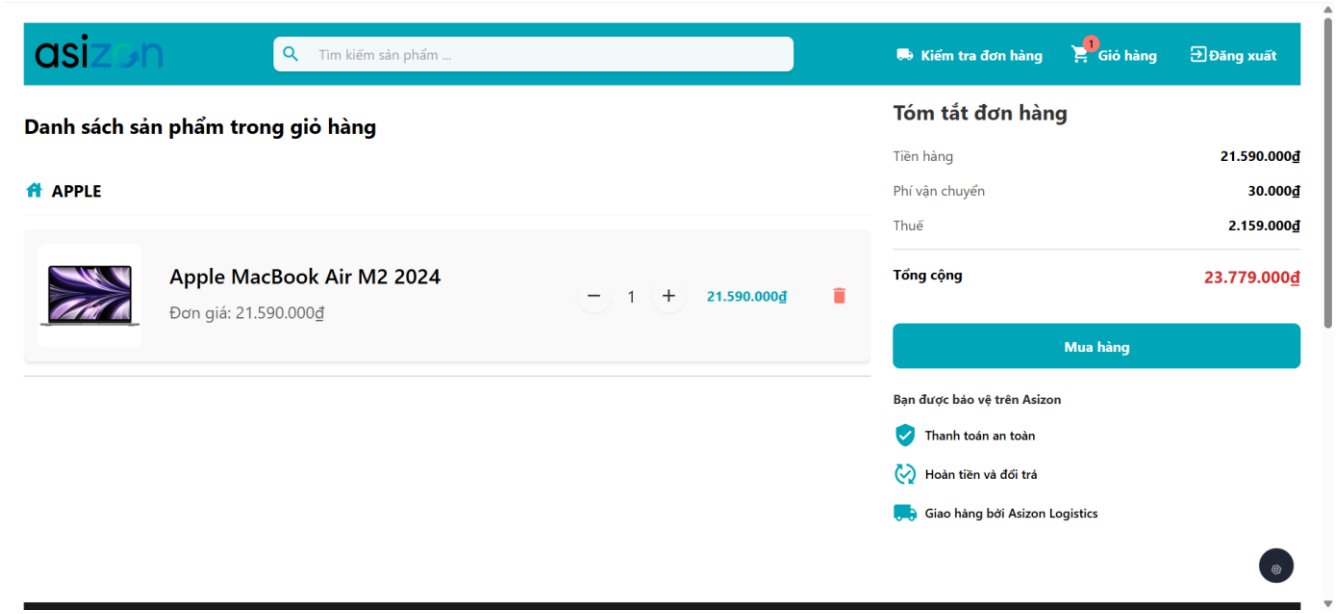
Hình 13: Điền thông tin mua hàng



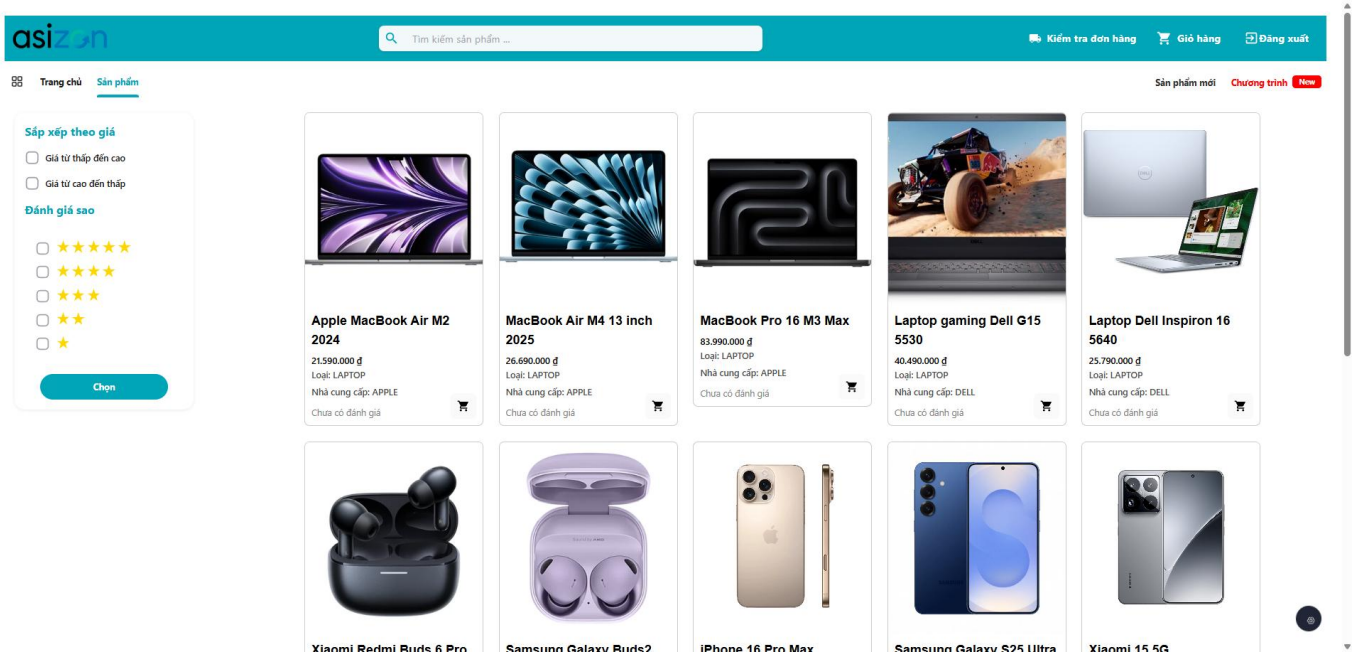
Hình 14: Form nhập thông tin giao hàng



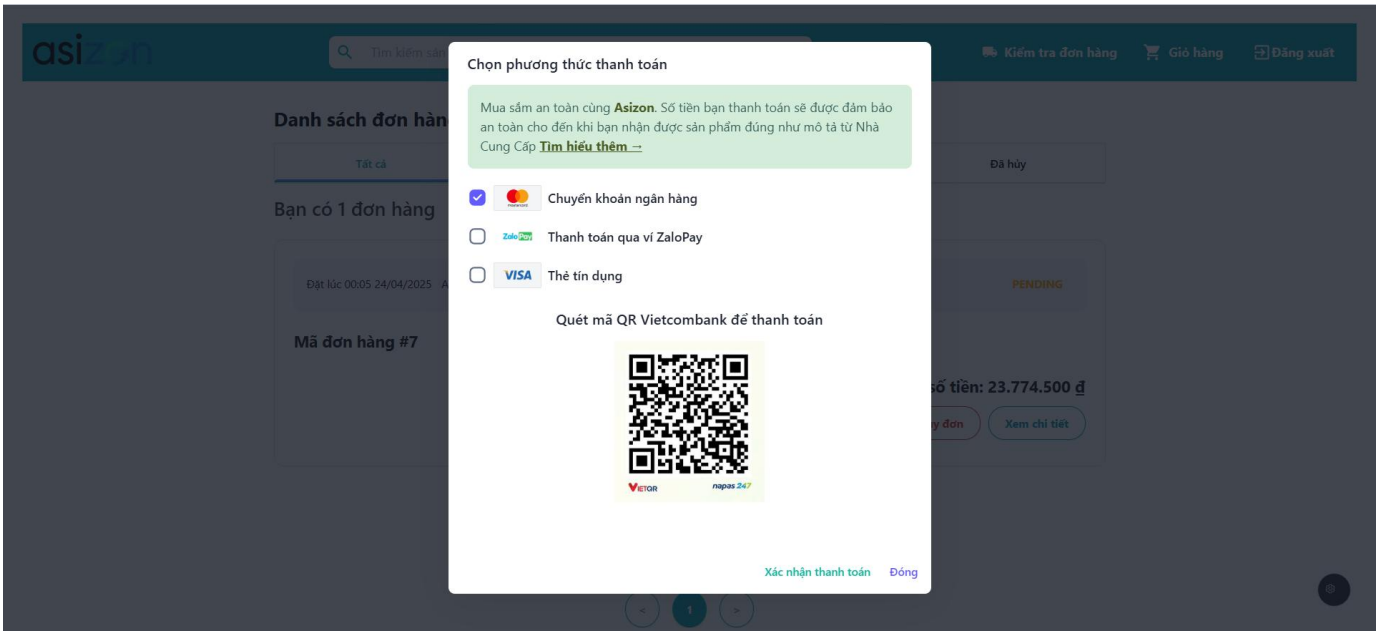
Hình 15: Quản lý đơn hàng



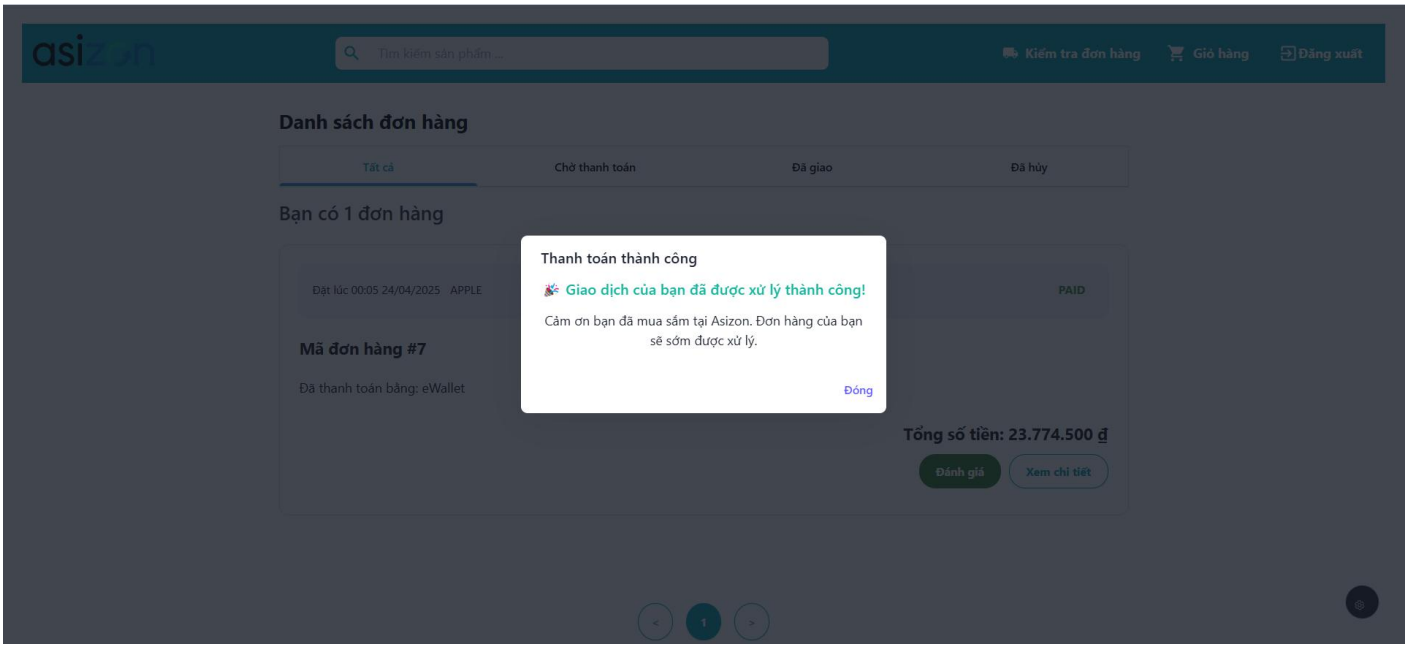
Hình 16: Quản lý giỏ hàng



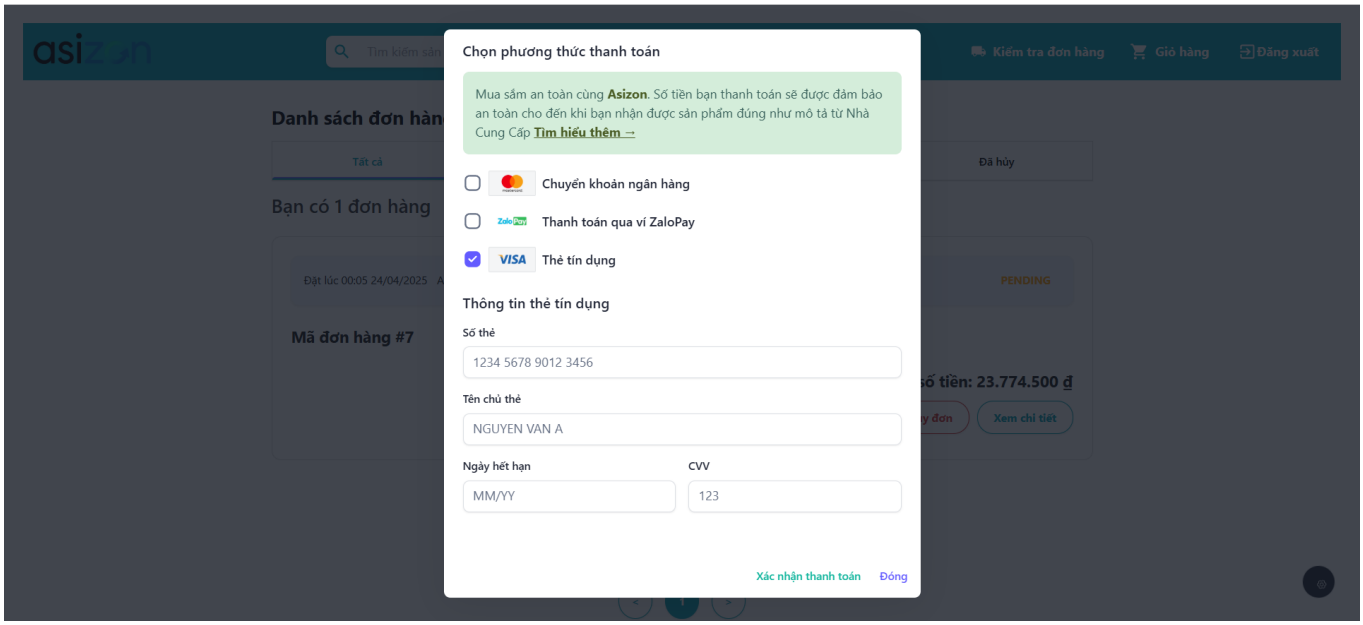
Hình 17: Trang sản phẩm



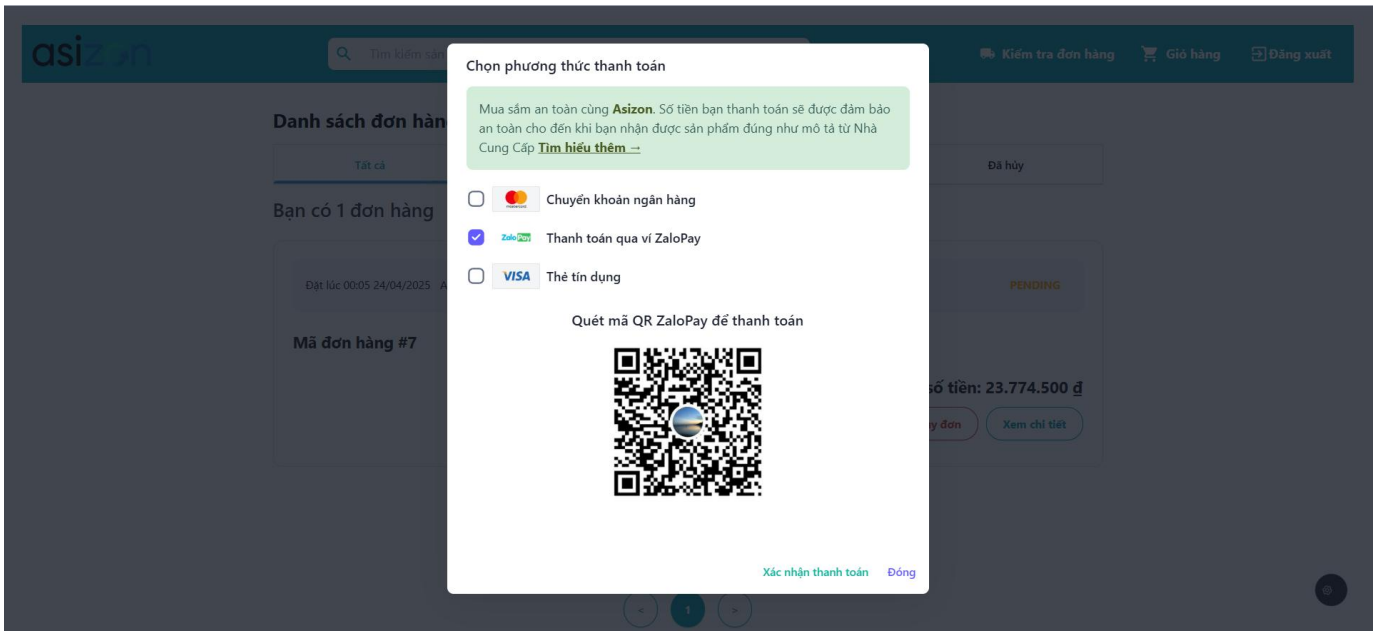
Hình 18: Thanh toán ngân hàng



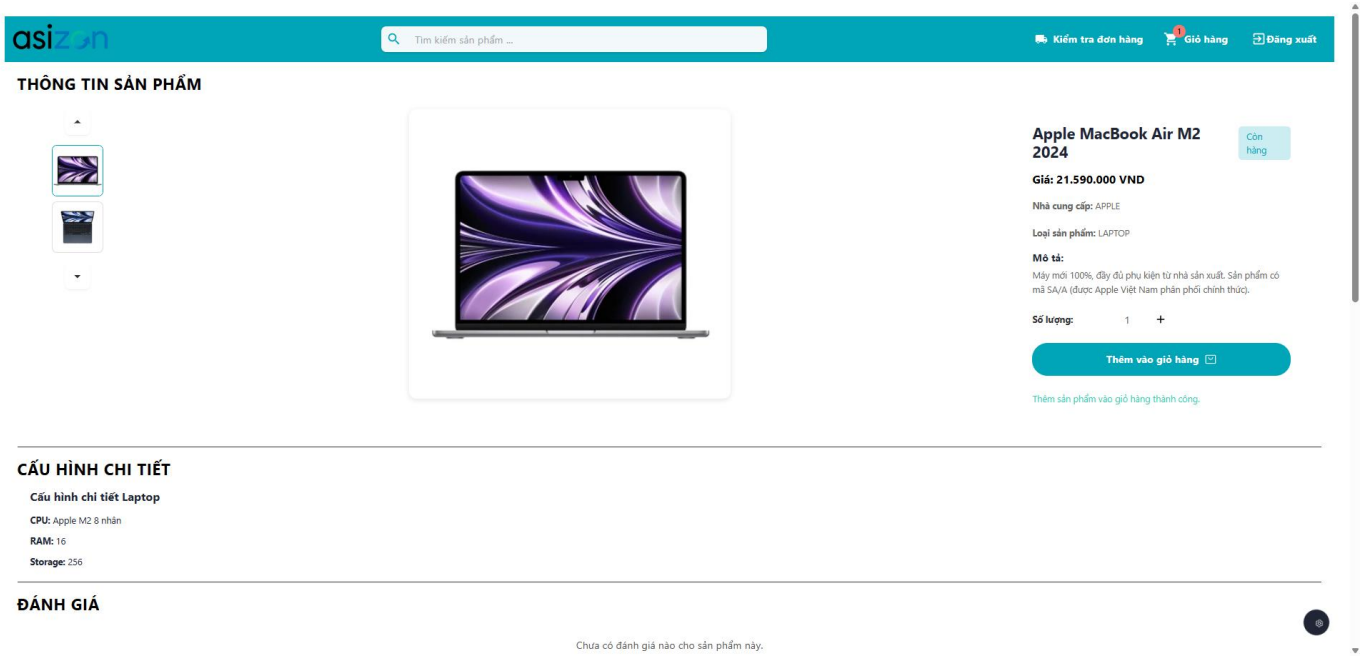
Hình 19: Thanh toán thành công



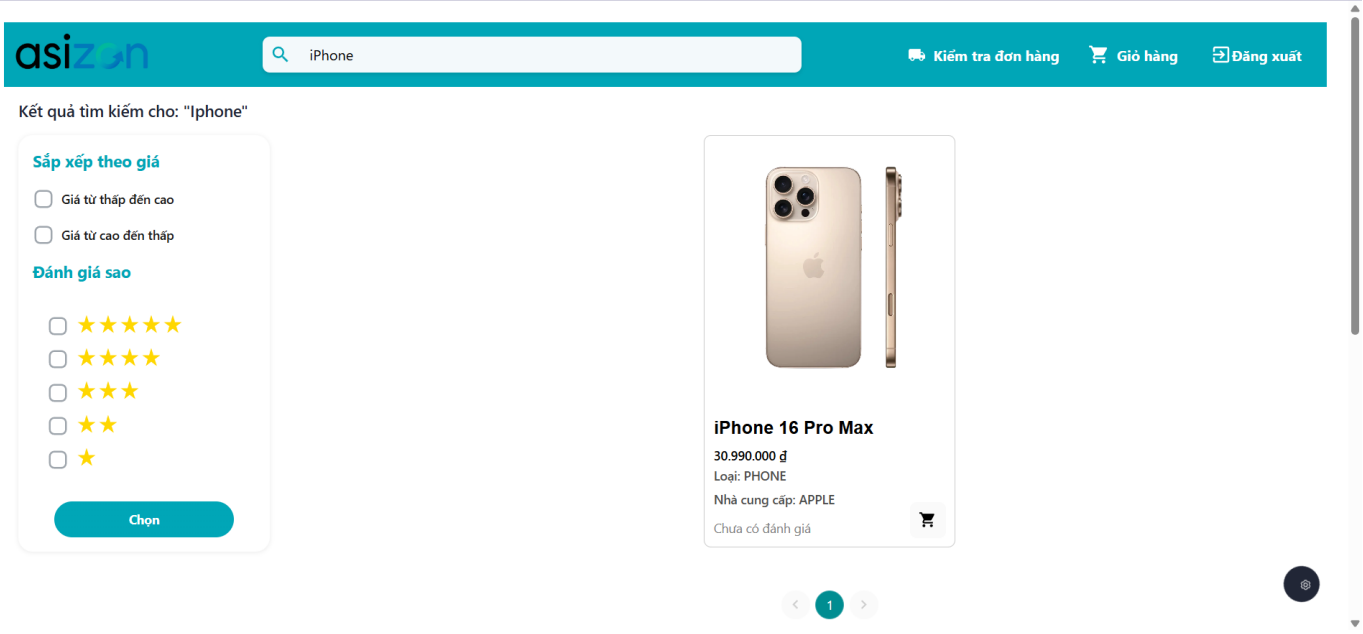
Hình 20: Thanh toán thẻ visa



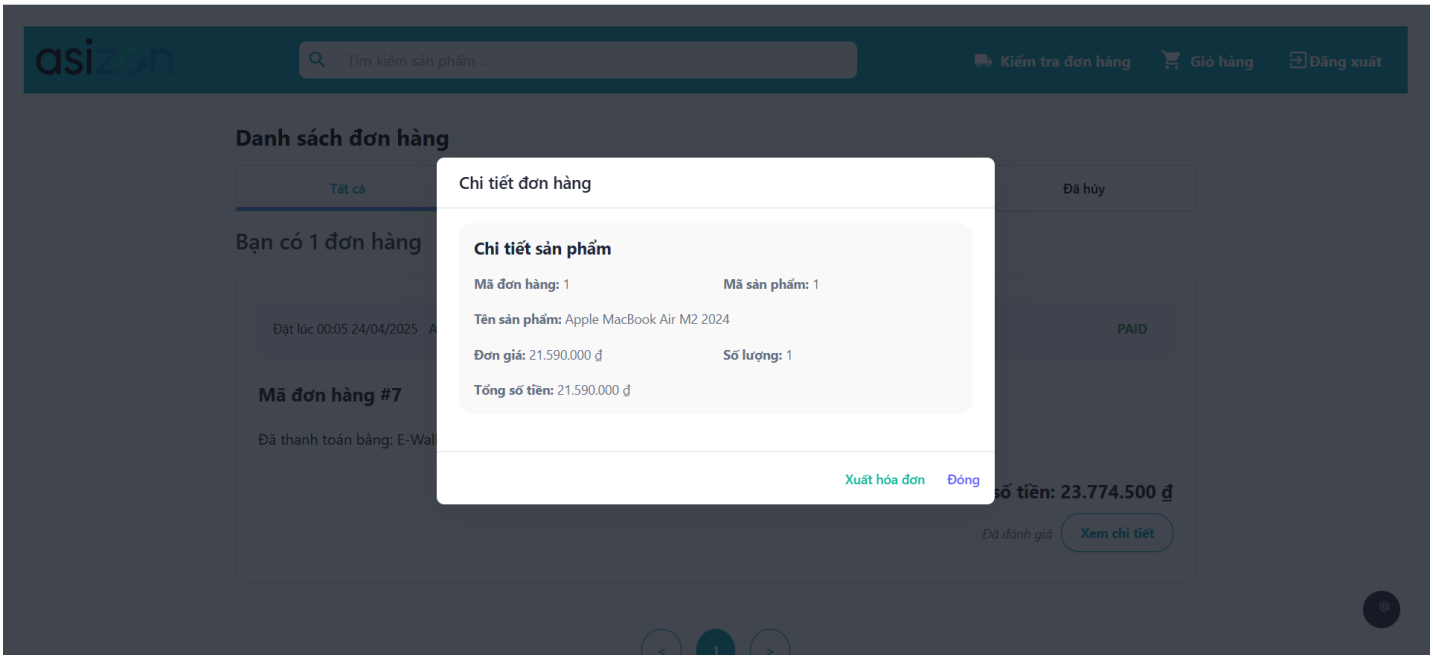
Hình 21: Thanh toán qua zalopay



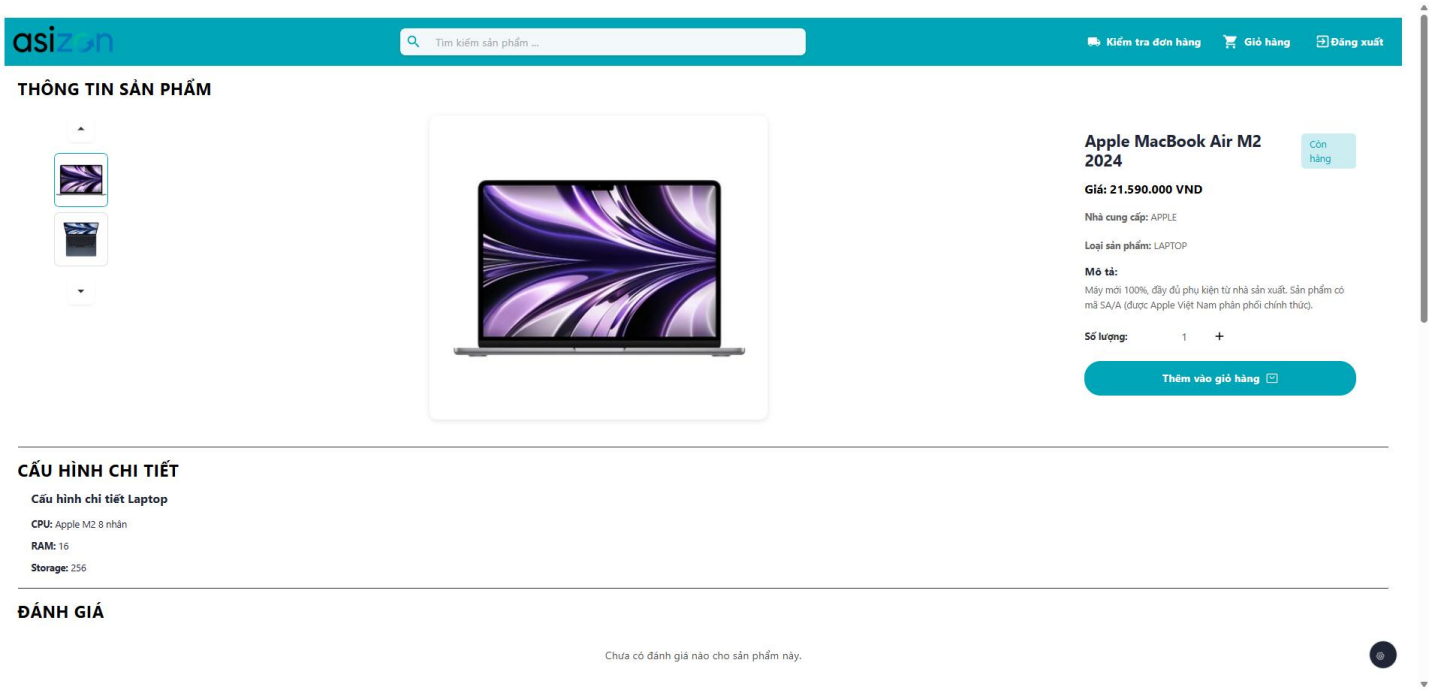
Hình 22 : Thêm sản phẩm vào giỏ hàng



Hình 23: Tìm kiếm sản phẩm

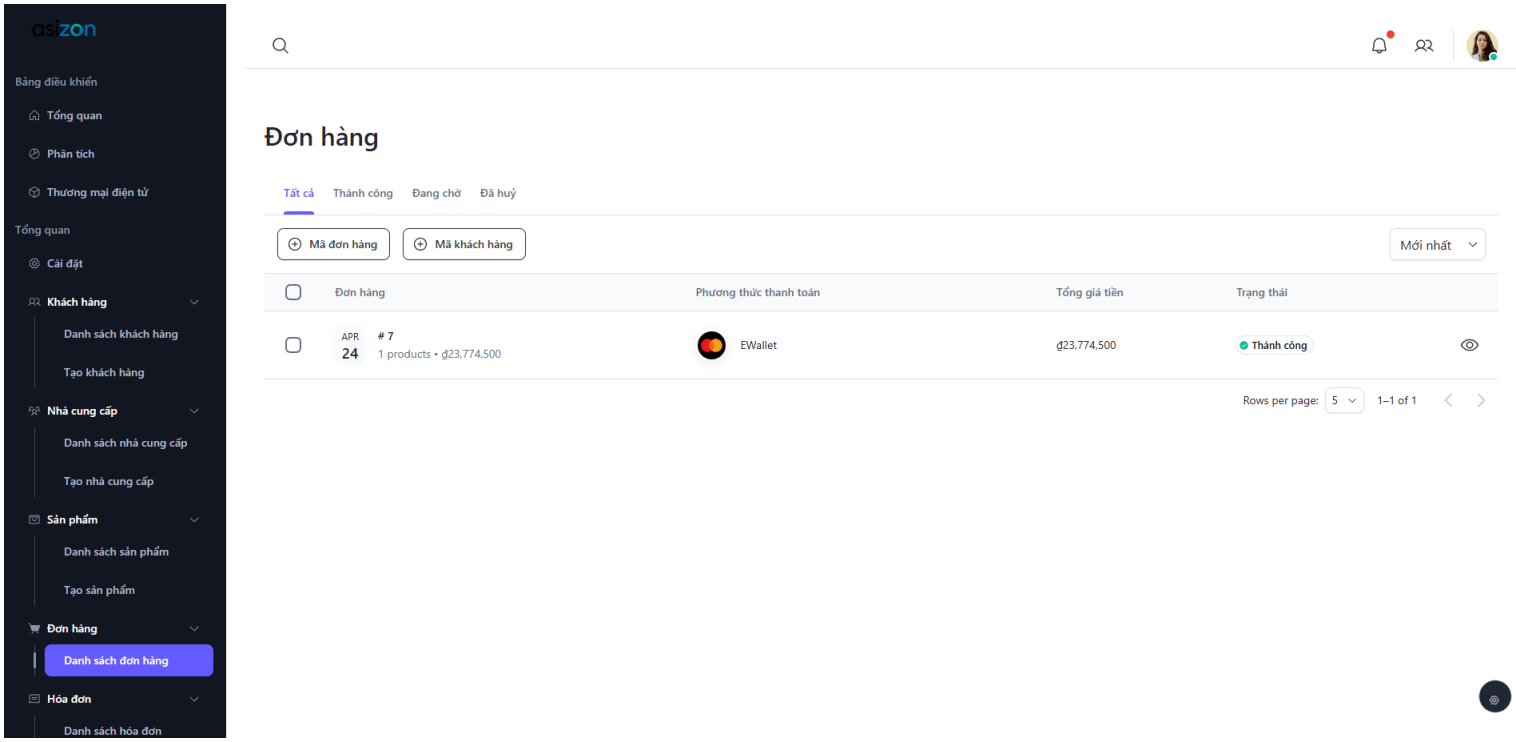


Hình 24: Chi tiết đơn hàng

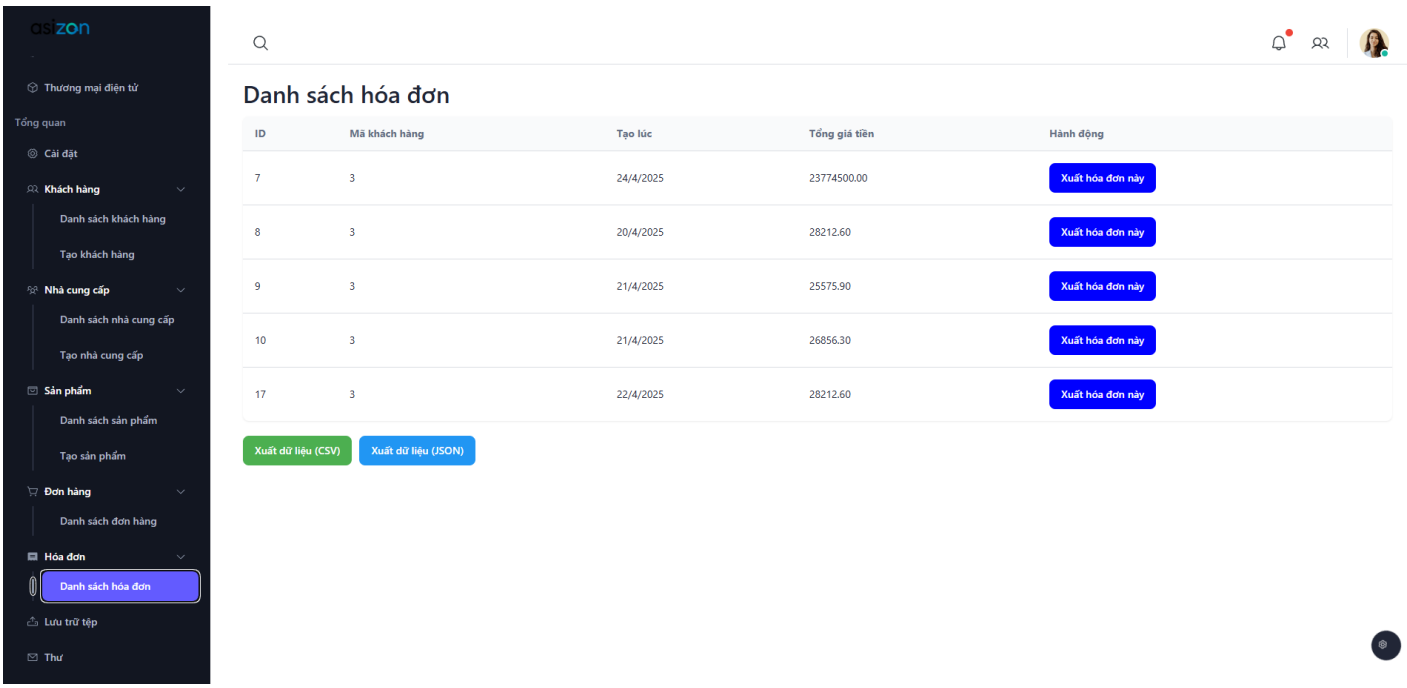


Hình 25: Chi tiết sản phẩm

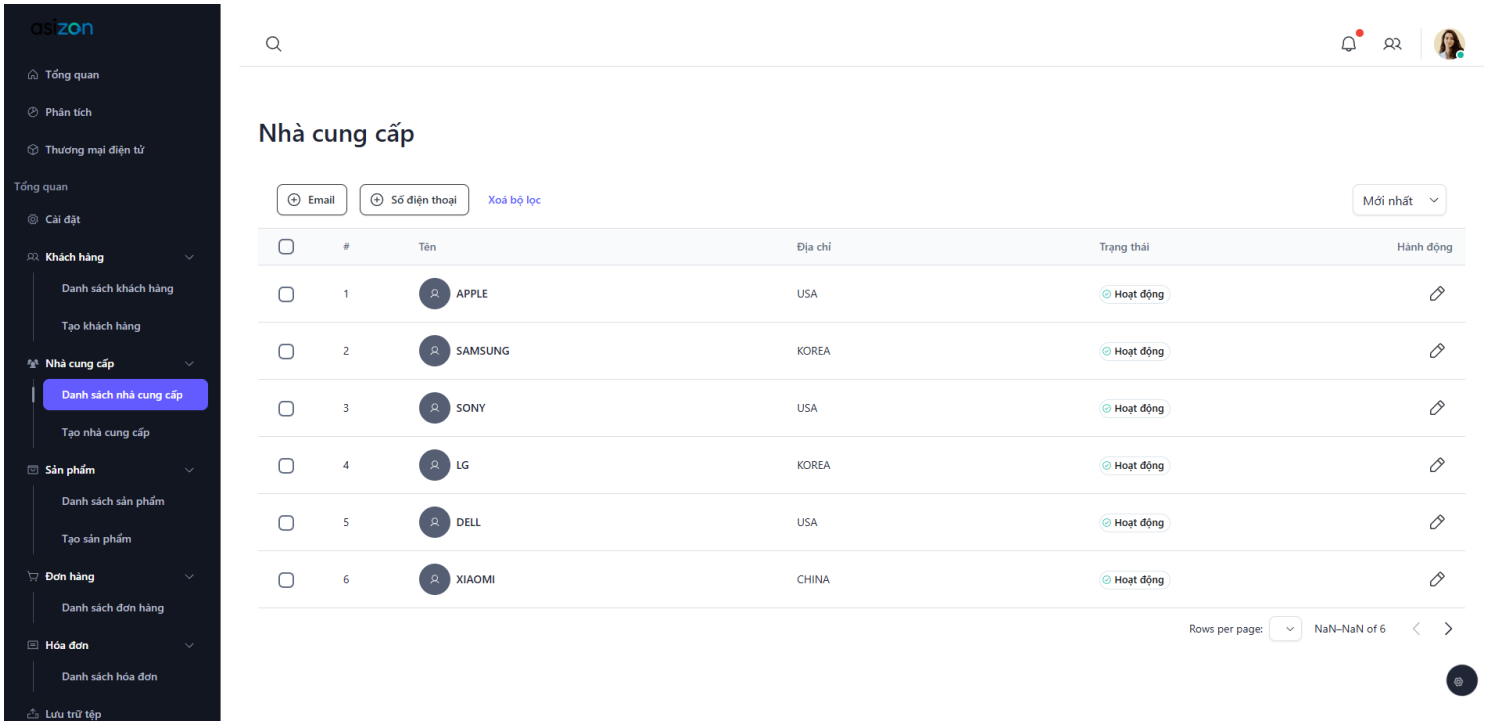
5.2 Giao diện từ quản lý



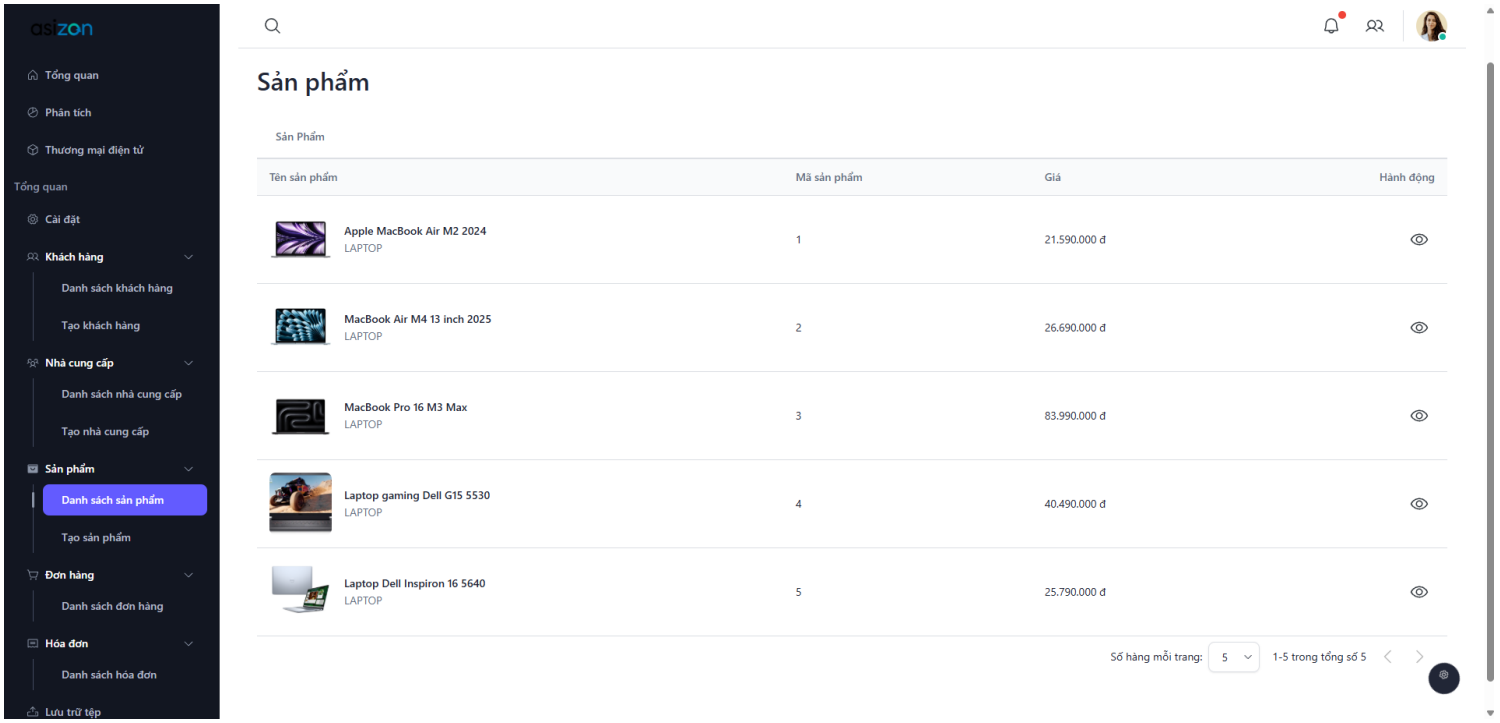
Hình 26: Quản lý đơn hàng



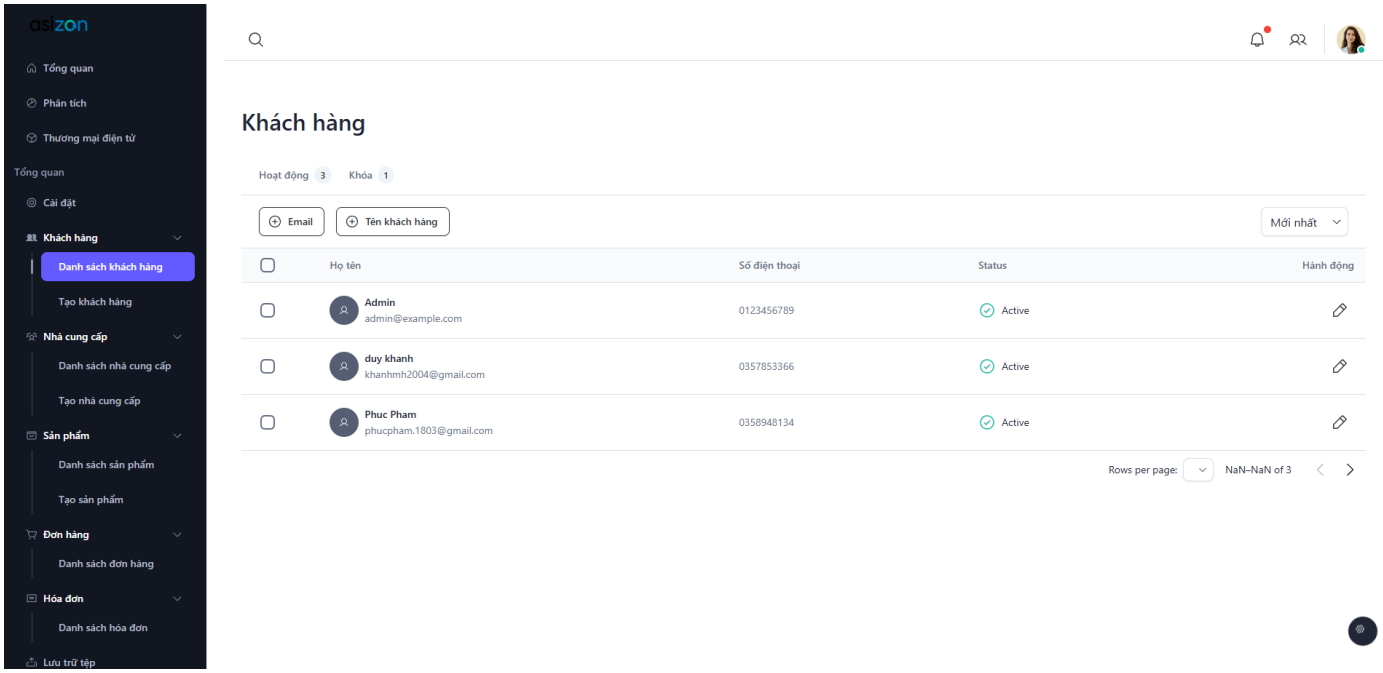
Hình 27: Quản lý hoá đơn



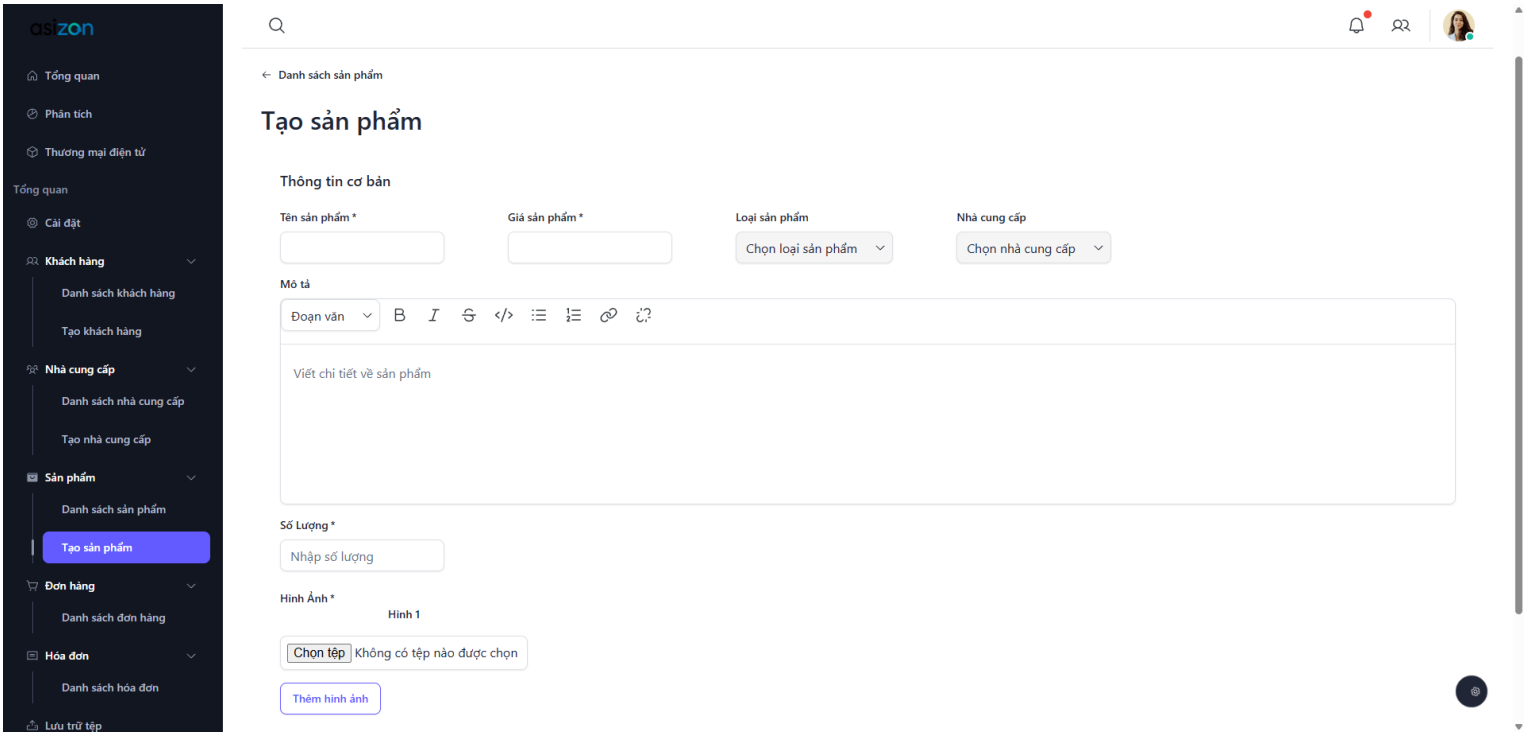
Hình 28: Quản lý nhà cung cấp



Hình 29: Quản lý sản phẩm



Hình 30: Quản lý khách hàng



Hình 31: Tạo sản phẩm mới

TÀI LIỆU THAM KHẢO

1. Eric Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra, [2004], Head First Design Patterns, O'Reilly Media, Sebastopol.
2. Steven John Metsker, William C. Wake, [2006], Design Patterns in Java, Addison-Wesley, New Jersey.
3. Erich Gamma, John Vlissides, Ralph Johnson, Richard Helm, [1995], Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, Boston.
4. Christopher G. Lasater, [2007], Design Patterns, Worldware Publications, Texas.